PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : H04J 3/14, 3/24, H04L 12/56

A1

(11) International Publication Number: WO 92/19054

(43) International Publication Date: 29 October 1992 (29.10.92)

(21) International Application Number: PCT/US92/02995

(22) International Filing Date: 10 April 1992 (10.04.92)

(30) Priority data:
684,695 12 April 1991 (12.04.91) US

(71) Applicant: CONCORD COMMUNICATIONS, INC. [US/US]; 753 Forest Street, Marlboro, MA 01752 (US).

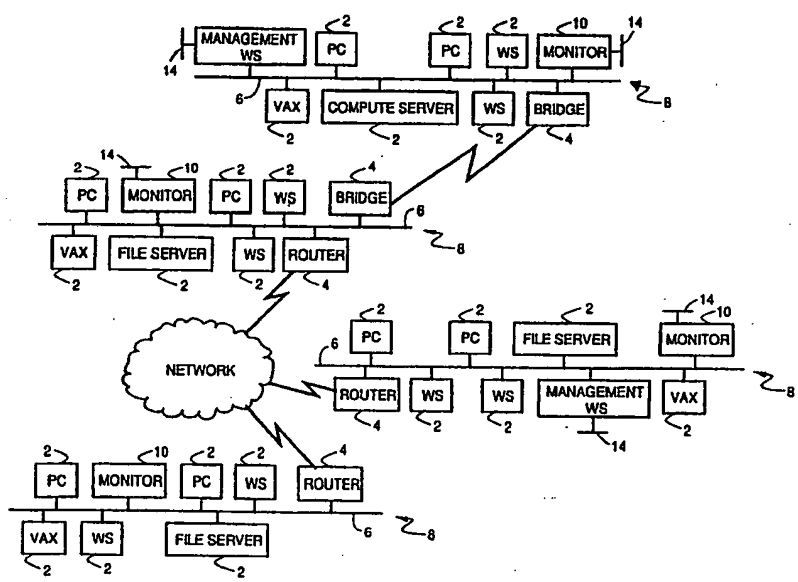
(72) Inventors: FERDINAND, Engel ; 21 Joseph Road, Northborough, MA 01532 (US). JONES, Kendall, S. ; 90 Boulder Road, Newton Center, MA 02159 (US). ROBERTSON, Kary ; 398 North Road, Bedford, MA 01739 (US). THOMPSON, David, M. ; 5127 243rd Road, Redmond, WA 98053 (US). WHITE, Gerard ; 133 Massapoag Road, Tyngsborough, MA 01879 (US).

(74) Agent: PRAHL, Eric, L.; Fish & Richardson, 225 Franklin Street, Boston, MA 02110-2804 (US).

(81) Designated States: AT (European patent), BE (European patent), CA, CH (European patent), DE (European patent), DK (European patent), ES (European patent), FR (European patent), GB (European patent), GR (European patent), IT (European patent), JP, LU (European patent), MC (European patent), NL (European patent), SE (European patent).

Published
With international search report.

(54) Title: NETWORK MONITORING



(57) Abstract

Monitoring is done of communications which occur in a network of nodes (2), each communication being effected by a transmission of one or more packets among two or more communicating nodes (2), each communication complying with a predefined communication protocol selected from among protocols available in the network. The contents of packets are detected passively and in real time, communication information (130, 152, 178) associated with multiple protocols is derived from the packet contents.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	ES	Spain	MG	Madagascar
AU	Australia	FI	Finland	ML	Mali
BB	Barbados	FR	France	MN	Mongolia
BE	Belgium	GA	Gabon	MR	Mauritania
BF	Burkina Faso	GB	United Kingdom	MW	Malawi
BG	Bulgaria	GN	Guinea	NL	Netherlands
BJ	Benin	GR	Greece	NO	Norway
BR	Brazil	HU	Hungary	PL	Poland
CA	Canada	IT	Italy	RO	Romania
CF	Central African Republic	JP	Japan	RU	Russian Federation
CG	Congo	KP	Democratic People's Republic of Korea	SD	Sudan
CH	Switzerland			SE	Sweden
CI	Côte d'Ivoire	KR	Republic of Korea	SN	Senegal
CM	Cameroon	LI	Liechtenstein	SU	Soviet Union
CS	Czechoslovakia	LK	Sri Lanka	TD	Chad
DE	Germany	LU	Luxembourg	TG	Togo
DK	Denmark	MC	Monaco	US	United States of America

NETWORK MONITORING

Background of the Invention

The invention relates to monitoring and managing communication networks for computers.

Todays computer networks are large complex systems with many components from a large variety of vendors. These networks often span large geographic areas ranging from a campus-like setting to world wide networks. While the network itself can be used by many different types of organizations, the purpose of these networks is to move information between computers. Typical applications are electronic mail, transaction processing, remote database, query, and simple file transfer. Usually, the organization that has installed and is running the network needs the network to be running properly in order to operate its business. Since these networks are complex systems, there are various controls provided by the different equipment to control and manage the network. Network management is the task of planning, engineering, securing and operating a network.

To manage the network properly, the Network Manager has some obvious needs. First, the Network Manager must trouble shoot problems. As the errors develop in a running network, the Network Manager must have some tools that notify him of the errors and allow him to diagnose and repair these errors. Second, the Network Manager needs to configure the network in such a manner that the network loading characteristics provide the best service possible for the network users. To do this the Network Manager must have tools that allow him visibility into access patterns, bottlenecks and general loading. With such data, the Network Manager can reconfigure the network components for better service.

There are many different components that need to be managed in the network. These elements can be, but

are not limited to: routers, bridges, PC's, workstations, minicomputers, supercomputers, printers, file servers, switches and pbx's. Each component provides a protocol for reading and writing the management variables in the machine. These variables are usually defined by the component vendor and are usually referred to as a Management Information Base (MIB). There are some standard MIB's, such as the IETF (Internet Engineering Task Force) MIB I and MIB II standard definitions. Through the reading and writing of MIB variables, software in other computers can manage or control the component. The software in the component that provides remote access to the MIB variables is usually called an agent. Thus, an individual charged with the responsibility of managing a large network often will use various tools to manipulate the MIB's of various agents on the network.

Unfortunately, the standards for accessing MIBs are not yet uniformly provided nor are the MIB definitions complete enough to manage an entire network. The Network Manager must therefore use several different types of computers to access the agents in the network. This poses a problem, since the errors occurring on the network will tend to show up in different computers and the Network Manager must therefore monitor several different screens to determine if the network is running properly. Even when the Network Manager is able to accomplish this task, the tools available are not sufficient for the Network Manager to function properly.

Furthermore, there are many errors and loadings on the network that are not reported by agents. Flow control problems, retransmissions, on-off segment loading, network capacities and utilizations are some of the types of data that are not provided by the agents.

Simple needs like charging each user for actual network usage are impossible.

Summary of the Invention

In general, in one aspect, the invention features monitoring communications which occur in a network of nodes, each communication being effected by a transmission of one or more packets among two or more communicating nodes, each communication complying with a predefined communication protocol selected from among protocols available in the network. The contents of packets are detected passively and in real time, communication information associated with multiple protocols is derived from the packet contents.

Preferred embodiments of the invention include the following features. The communication information derived from the packet contents is associated with multiple layers of at least one of the protocols.

In general, in another aspect, the invention features monitoring communication dialogs which occur in a network of nodes, each dialog being effected by a transmission of one or more packets among two or more communicating nodes, each dialog complying with a predefined communication protocol selected from among protocols available in the network. Information about the states of dialogs occurring in the network and which comply with different selected protocols available in the network is derived from the packet contents.

Preferred embodiments of the invention include the following features. A current state is maintained for each dialog, and the current state is updated in response to the detected contents of transmitted packets. For each dialog, a history of events is maintained based on information derived from the contents of packets, and the history of events is analyzed to derive information about the dialog. The analysis of the history includes

counting events and gathering statistics about events. The history is monitored for dialogs which are inactive, and dialogs which have been inactive for a predetermined period of time are purged. For example, the current state is updated to data state in response to observing the transmission of at least two data related packets from each node. Sequence numbers of data related packets stored in the history of events are analyzed and retransmissions are detected based on the sequence numbers. The the current state is updated based on each new packet associated with the dialog; if an updated current state cannot be determined, information about prior packets associated with the dialog is consulted as an aid in updating the state. The history of events may be searched to identify the initiator of a dialog.

The full set of packets associated with a dialog up to a point in time completely define a true state of the dialog at that point in time, and the step of updating the current state in response to the detected contents of transmitted packets includes generating a current state (e.g., "unknown") which may not conform to the true state. The current state may be updated to the true state based on information about prior packets transmitted in the dialog.

Each communication may involve multiple dialogs corresponding to a specific protocol. Each protocol layer of the communication may be parsed and analyzed to isolate each dialog and statistics may be kept for each dialog. The protocols may include a connectionless-type protocol in which the state of a dialog is implicit in transmitted packets, and the step of deriving information about the states of dialogs includes inferring the states of the dialogs from the packets. Keeping statistics for protocol layers may be temporarily suspended when parsing

and statistics gathering is not rapid enough to match the rate of packets to be parsed.

In general, in another aspect, the invention features monitoring the operation of the network with respect to specific items of performance during normal operation, generating a model of the network based on the monitoring, and setting acceptable threshold levels for the specific items of performance based on the model. In preferred embodiments, the operation of the network is monitored with respect to the specific items of performance during periods which may include abnormal operation.

In general, in another aspect, the invention features the combination of a monitor connected to the network medium for passively, and in real time, monitoring transmitted packets and storing information about dialogs associated with the packets, and a workstation for receiving the information about dialogs from the monitor and providing an interface to a user. In preferred embodiments, the workstation includes means for enabling a user to observe events of active dialogs.

In general, in another aspect, the invention features apparatus for monitoring packet communications in a network of nodes in which communications may be in accordance with multiple protocols. The apparatus includes a monitor connected to a communication medium of the network for passively, and in real time, monitoring transmitted packets of different protocols and storing information about communications associated with the packets, the communications being in accordance with different protocols, and a workstation for receiving the information about the communciations from the monitor and providing an interface to a user. The monitor and the workstation include means for relaying the information about multiple protocols with respect to communication in

the different protocols from the monitor to the workstation in accordance with a single common network management protocol.

In general, in another aspect, the invention features diagnosing communication problems between two nodes in a network of nodes interconnected by links. The operation of the network is monitored with respect to specific items of performance during normal operation. A model of normal operation of the network is generated based on the monitoring. Acceptable threshold levels are set for the specific items of performance based on the model. The operation of the network is monitored with respect to the specific items of performance during periods which may include abnormal operation. When abnormal operation of the network with respect to communication between the two nodes is detected, the problem is diagnosed by separately analyzing the performance of each of the nodes and each of the links connecting the two nodes to isolate the abnormal operation.

In general, in another aspect, the invention features a method of timing the duration of a transaction of interest occurring in the course of communication between nodes of a network, the beginning of the transaction being defined by the sending of a first packet of a particular kind from one node to the other, and the end of the transaction being defined by the sending of another packet of a particular kind between the nodes. In the method, packets transmitted in the network are monitored passively and in real time. The beginning time of the transaction is determined based on the appearance of the first packet. A determination is made of when the other packet has been transmitted. The timing of the duration of the transaction is ended upon the appearance of the other packet.

In general, in another aspect, the invention features, tracking node address to node name mappings in a network of nodes of the kind in which each node has a possibly nonunique node name and a unique node address within the network and in which node addresses can be assigned and reassigned to node names dynamically using a name binding protocol message incorporated within a packet. In the method, packets transmitted in the network are monitored, and a table linking node names to node addresses is updated based on information contained in the name binding protocol messages in the packets.

One advantage of the invention is that it enables a network manager to passively monitor multi-protocol networks at multiple layers of the communications. In addition, it organizes and presents network performance statistics in terms of dialogs which are occurring at any desired level of the communication. This technique of organizing and displaying network performance statistics provides an effective and useful view of network performance and facilitates a quick diagnosis of network problems.

Other advantages and features will become apparent from the following description of the preferred embodiment and from the claims.

Description of the Preferred Embodiments

Fig. 1 is a block diagram of a network;

Fig. 2 shows the layered structure of a network communication and a protocol tree within that layered environment;

Fig. 3 illustrates the structure of an ethernet/IP/TCP packet;

Fig. 4 illustrates the different layers of a communication between two nodes;

Fig. 5 shows the software modules within the Monitor;

Fig. 6 shows the structure of the Monitor software in terms of tasks and intertask communication mechanisms;

Figs. 7a-c show the STATS data structures which store performance statistics relating to the the data link layer;

Fig. 8 is a event/state table describing the operation of the state machine for a TCP connection;

Fig. 9a is a history data structure that is identified by a pointer found in the appropriate dialog statistics data within STATS;

Fig. 9b is a record from the history table;

Fig. 10 is a flow diagram of the Look_for_Data_State routine;

Fig. 11 is a flow diagram of the Look_for_Initiator routine that is called by the Look_for_Data_State routine;

Fig. 12 is a flow diagram of the Look_for_Retransmission routine which is called by the Look_at_History routine;

Fig. 13 is a diagram of the major steps in processing a frame through the Real Time Parser (RTP);

Fig. 14 is a diagram of the major steps in the processing a statistics threshold event;

Fig. 15 is a diagram of the major steps in the processing of a database update;

Fig. 16 is a diagram of the major steps in the processing of a monitor control request;

Fig. 17 is a logical map of the network as displayed by the Management Workstation;

Fig. 18 is a basic summary tool display screen;

Fig. 19 is a protocol selection menu that may be invoked through the summary tool display screen;

Figs. 20a-g are examples of the statistical variables which are displayed for different protocols;

Fig. 21 is an example of information that is displayed in the dialogs panel of the summary tool display screen;

Fig. 22 is a basic data screen presenting a rate values panel, a count values panel and a protocols seen panel;

Fig. 23 is a traffic matrix screen;

Fig. 24 is a flow diagram of the algorithm for adaptively establishing network thresholds based upon actual network performance;

Fig. 25 is a simple multi-segment network;

Fig. 26 is a flow diagram of the operation of the diagnostic analyzer algorithm;

Fig. 27 is a flow diagram of the source node analyzer algorithm;

Fig. 28 is a flow diagram of the sink node analyzer algorithm;

Fig. 29 is a flow diagram of the link analysis logic;

Fig. 30 is a flow diagram of the DLL problem checking routine;

Fig. 31 is a flow diagram of the IP problem checking routine;

Fig. 32 is a flow diagram of the IP link component problem checking routine;

Fig. 33 is a flow diagram of the DLL link component problem checking routine;

Fig. 34 shows the structure of the event timing database;

Fig. 35 is a flow diagram of the operation of the event timing module (ETM) in the Network Monitor;

Fig. 36 is a network which includes an Appletalk® segment;

Fig. 37 is a Name Table that is maintained by the Address Tracking Module (ATM);

Fig. 38 is a flow diagram of the operation of the ATM; and

Fig. 39 is a flow diagram of the operation of the ATM.

Also attached hereto before the claims are the following appendices:

Appendix I identifies the SNMP MIB subset that is supported by the Monitor and the Management Workstation (2 pages);

Appendix II defines the extension to the standard MIB that are supported by the Monitor and the Management Workstation (25 pages);

Appendix III is a summary of the protocol variables for which the Monitor gathers statistics and a brief description of the variables, where appropriate (17 pages);

Appendix IV is a list of the Summary Tool Values Display Fields with brief descriptions (2 pages); and

Appendix V is a description of the actual screens for the Values Tool (34 pages).

<u>Structure and Operation</u>

<u>The Network</u>:

A typical network, such as the one shown in Fig. 1, includes at least three major components, namely, network nodes 2, network elements 4 and communication lines 6. Network nodes 2 are the individual computers on the network. They are the very reason the network exists. They include but are not limited to workstations (WS), personal computers (PC), file servers (FS), compute servers (CS) and host computers (e.g., a VAX), to name but a few. The term server is often used as though it was different from a node, but it is, in fact, just a node providing special services.

In general, network elements 4 are anything that participate in the service of providing data movement in

a network, i.e., providing the basic communications. They include, but are not limited to, LAN's, routers, bridges, gateways, multiplexors, switches and connectors. Bridges serve as connections between different network segments. They keep track of the nodes which are connected to each of the segments to which they are connected. When they see a packet on one segment that is addressed to a node on another of their segments, they grab the packet from the one segment and transfer it to the proper segment. Gateways generally provide connections between different network segments that are operating under different protocols and serve to convert communications from one protocol to the other. Nodes send packets to routers so that they may be directed over the appropriate segments to the intended destination node.

Finally, network or communication lines 6 are the components of the network which connect nodes 2 and elements 4 together so that communicatons between nodes 2 may take place. They can be private lines, satellite lines or Public Carrier lines. They are expensive resources and are usually managed as separate entities. Often networks are organized into segments 8 that are connected by network elements 4. A segment 8 is a section of a LAN connected at a physical level (this may include repeaters). Within a segment, no protocols at layers above the physical layer are needed to enable signals from two stations on the same segment to reach each other (i.e., there are no routers, bridges, gateways...).

<u>The Network Monitor and the Management Workstation</u>:

In the described embodiment, there are two basic elements to the monitoring system which is to be described, namely, a Network Monitor 10 and a Management

Workstation 12. Both elements interact with each other over the local area network (LAN).

Network Monitor 10 (referred to hereinafter simply as Monitor 10) is the data collection module which is attached to the LAN. It is a high performance real time front end processor which collects packets on the network and performs some degree of analysis to search for actual or potential problems and to maintain statistical information for use in later analysis. In general, it performs the following functions. It operates in a promiscuous mode to capture and analyze all packets on the segment and it extracts all items of interest from the frames. It generates alarms to notify the Management Workstation of the occurence of significant events. It receives commands from the Management Workstation, processes them appropriately and returns responses.

Management Workstation 12 is the operator interface. It collects and presents troubleshooting and performance information to the user. It is based on the SunNet Manager (SNM) product and provides a graphical network-map-based interface and sophisticated data presentation and analysis tools. It receives information from Monitor 10, stores it and displays the information in various ways. It also instructs Monitor 10 to perform certain actions. Monitor 10, in turn, sends responses and alarms to Management Workstation 12 over either the primary LAN or a backup serial link 14 using SNMP with the MIB extensions defined later.

These devices can be connected to each other over various types of networks and are not limited to connections over a local area network. As indicated in Fig. 1, there can be multiple Workstations 12 as well as multiple Monitors 10.

Before describing these components in greater detail, background information will first be reviewed

regarding communication protocols which specify how communications are conducted over the network and regarding the structure of the packets.

The Protocol Tree:

As shown in Fig. 2, communication over the network is organized as a series of layers or levels, each one built upon the next lower one, and each one specified by one or more protocols (represented by the boxes). Each layer is responsible for handling a different phase of the communication between nodes on the network. The protocols for each layer are defined so that the services offered by any layer are relatively independent of the services offered by the neighbors above and below. Although the identities and number of layers may differ depending on the network (i.e., the protocol set defining communication over the network), in general, most of them share a similar structure and have features in common.

For purposes of the present description, the Open Systems Interconnection (OSI) model will be presented as representative of structured protocol architectures. The OSI model, developed by the International Organization for Standardization, includes seven layers. As indicated in Fig. 2, there is a physical layer, a data link layer (DLL), a network layer, a transport layer, a session layer, a presentation layer and an application layer, in that order. As background for what is to follow, the function of each of these layers will be briefly described.

The physical layer provides the physical medium for the data transmission. It specifies the electrical and mechanical interfaces of the network and deals with bit level detail. The data link layer is responsible for ensuring an error-free physical link between the communicating nodes. It is responsible for creating and recognizing frame boundaries (i.e., the boundaries of the

packets of data that are sent over the network.) The network layer determines how packets are routed within the network. The transport layer accepts data from the layer above it (i.e., the session layer), breaks the packets up into smaller units, if required, and passes these to the network layer for transmission over the network. It may insure that the smaller pieces all arrive properly at the other end. The session layer is the user's interface into the network. The user must interface with the session layer in order to negotiate a connection with a process in another machine. The presentation layer provides code conversion and data reformatting for the user's application. Finally, the application layer selects the overall network service for the user's application.

Fig. 2 also shows the protocol tree which is implemented by the described embodiment. A protocol tree shows the protocols that apply to each layer and it identifies by the tree structure which protocols at each layer can run "on top of" the protocols of the next lower layer. Though standard abbreviations are used to identify the protocols, for the convenience of the reader, the meaning of the abbreviations are as follows:

ARP	Address Resolution Protocol
ETHERNET	Ethernet Data Link Control
FTP	File Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
LLC	802.2 Logical Link Control
MAC	802.3 CSMA/CD Media Access Control
NFS	Network File System
NSP	Name Server Protocol
RARP	Reverse Address Resolution Protocol
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol

TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol

Two terms are commonly used to describe the protocol tree, namely, a protocol stack and a protocol family (or suite). A protocol stack generally refers to the underlying protocols that are used when sending a message over a network. For example, FTP/TCP/IP/LLC is a protocol stack. A protocol family is a loose association of protocols which tend to be used on the same network (or derive from a common source). Thus, for example, the TCP/IP family includes IP, TCP, UDP, ARP, TELNET and FTP. The Decnet family includes the protocols from Digital Equipment Corporation. And the SNA family includes the protocols from IBM.

<u>The Packet</u>:

The relevant protocol stack defines the structure of each packet that is sent over the network. Fig. 3, which shows an TCP/IP packet, illustrates the typical structure of a packet. In general, each level of the protocol stack takes the data from the next higher level and adds header information to form a protocol data unit (PDU) which it passes to the next lower level. That is, as the data from the application is passed down through the protocol layers in preparation for transmission over the network, each layer adds its own information to the data passed down from above until the complete packet is assembled. Thus, the structure of a packet ressembles that of an onion, with each PDU of a given layer wrapped within the PDU of the adjacent lower level.

At the ethernet level, the PDU includes a destination address (DEST MAC ADDR), a source address (SRC MAC ADDR), a type (TYPE) identifying the protocol which is running on top of this layer, and a DATA field for the PDU from the IP layer.

Like the ethernet packet, the PDU for the IP layer includes an IP header plus a DATA field. The IP header includes a type field (TYPE) for indicating the type of service, a length field (LGTH) for specifying the total length of the PDU, an identification field (ID), a protocol field (PROT) for identifying the protocol which is running on top of the IP layer (in this case, TCP), a source address field (SRC ADDR) for specifying the IP address of the sender, a destination address field (DEST ADDR) for specifying the IP address of the destination node, and a DATA field.

The PDU built by the TCP protocol also consists of a header and the data passed down from the next higher layer. In this case the header includes a source port field (SRC PORT) for specifying the port number of the sender, a destination port field (DEST PORT) for specifying the port number of the destination, a sequence number field (SEQ NO.) for specifying the sequence number of the data that is being sent in this packet, and an acknowledgment number field (ACK NO.) for specifying the number of the acknowledgment being returned. It also includes bits which identify the packet type, namely, an acknowledgment bit (ACK), a reset connection bit (RST), a synchronize bit (SYN), and a no more data from sender bit (FIN). There is also a window size field (WINDOW) for specifying the size of the window being used.

<u>The Concept of a Dialog</u>:

The concept of a dialog is used throughout the following description. As will become apparent, it is a concept which provides a useful way of conceptualizing, organizing and displaying information about the performance of a network - for any protocol and for any layer of the multi-level protocol stack.

As noted above, the basic unit of information in communication is a packet. A packet conveys meaning

between the sender and the receiver and is part of a larger framework of packet exchanges. The larger exchange is called a dialog within the context of this document. That is, a dialog is a communication between a sender and a receiver, which is composed of one or more packets being transmitted between the two. There can be multiple senders and receivers which can change roles. In fact, most dialogs involve exchanges in both directions.

Stated another way, a dialog is the exchange of messages and the associated meaning and state that is inherent in any particular exchange at any layer. It refers to the exchange between the peer entities (hardware or software) in any communication. In those situations where there is a layering of protocols, any particular message exchange could be viewed as belonging to multiple dialogs. For example, in Fig. 4 Nodes A and B are exchanging packets and are engaged in multiple dialogs. Layer 1 in Node A has a dialog with Layer 1 in Node B. For this example, one could state that this is the data link layer and the nature of the dialog deals with the message length, number of messages, errors and perhaps the guarantee of the delivery. Simultaneously, Layer n of Node A is having a dialog with Layer n of node B. For the sake of the example, one could state that this is an application layer dialog which deals with virtual terminal connections and response rates. One can also assume that all of the other layers (2 through n-1) are also having simultaneous dialogs.

In some protocols there are explicit primitives that deal with the dialog and they are generally referred to as connections or virtual circuits. However, dialogs exist even in stateless and connectionless protocols. Two more examples will be described to help clarify the concept further, one dealing with a connection oriented

protocol and the other dealing with a connectionless protocol.

In a typical connection oriented protocol, Node A sends a connection request (CR) message to Node B. The CR is an explicit request to form a connection. This is the start of a particular dialog, which is no different from the start of the connection. Nodes A and B could have other dialogs active simultaneously with this particular dialog. Each dialog is seen as unique. A connection is a particular type of dialog.

In a typical connectionless protocol, Node A sends Node B a message that is a datagram which has no connection paradigm, in fact, neither do the protocol(s) at higher layers. The application protocol designates this as a request to initiate some action. For example, a file server protocol such as Sun Microsystems' Network File System (NFS) could make a mount request. A dialog comes into existence once the communication between Nodes A and B has begun. It is possible to determine that communication has occurred and to determine the actions being requested. If in fact there exists more than one communication thread between Nodes A and B, then these would represent separate, different dialogs.

<u>Inside the Network Monitor</u>:

Monitor 10 includes a MIPS R3000 general purpose microprocessor (from MIPS Computer Systems, Inc.) running at 25 MHz. It is capable of providing 20 mips processing power. Monitor 10 also includes a 64Kbyte instruction cache and a 64Kbyte data cache, implemented by SRAM.

The major software modules of Monitor 10 are implemented as a mixture of tasks and subroutine libraries as shown in Fig. 5. It is organized this way so as to minimise the context switching overhead incurred during critical processing sequences. There is NO PREEMPTION of any module in the monitor subsystem. Each

module is cognizant of the fact that it should return control to the kernel in order to let other tasks run. Since the monitor subsystem is a closed environment, the software is aware of real time constraints.

Among the major modules which make up Monitor 10 is a real time kernel 20, a boot/load module 22, a driver 24, a test module 26, an SNMP Agent 28, a Timer module 30, a real time parser (RTP) 32, a Message Transport Module (MTM) 34, a statistics database (STATS) 36, an Event Manager (EM) 38, an Event Timing Module (ETM) 40 and a control module 42. Each of these will now be described in greater detail.

Real Time Kernel 20 takes care of the general housekeeping activities in Monitor 10. It is responsible for scheduling, handling intertask communications via queues, managing a potentially large number of timers, manipulating linked lists, and handling simple memory management.

Boot/Load Module 22, which is FProm based, enables Monitor 10 to start itself when the power is turned on in the box. It initializes functions such as diagnostics, and environmental initialization and it initiates down loading of the Network Monitor Software including program and configuration files from the Management Workstation. Boot/load module 22 is also responsible for reloading program and/or configuration data following internal error detection or on command from the Management Workstation. To accomplish down loading, boot/load module 22 uses the Trivial File Transfer Protocol (TFTP). The protocol stack used for loading is TFTP/UDP/IP/ethernet over the LAN and TFTP/UDP/IP/SLIP over the serial line.

Device Driver 24 manages the network controller hardware so that Monitor 10 is able to read and write packets from the network and it manages the serial

interface. It does so both for the purposes of monitoring traffic (promiscuous mode) and for the purposes of communicating with the Management Workstation and other devices on the network. The communication occurs through the network controller hardware of the physical network (e.g. Ethernet). The drivers for the LAN controller and serial line interface are used by the boot load module and the MTM. They provide access to the chips and isolate higher layers from the hardware specifics.

Test module 26 performs and reports results of physical layer tests (TDR, connectivity,...) under control of the Management Workstation. It provides traffic load information in response to user requests identifying the particular traffic data of interest. The load information is reported either as a percent of available bandwidth or as frame size(s) plus rate.

SNMP Agent 28 translates requests and information into the network management protocol being used to communicate with the Management Workstation, e.g., the Simple Network Management Protocol (SNMP).

Control Module 42 coordinates access to monitor control variables and performs actions necessary when these are altered. Among the monitor control variables which it handles are the following:

set reset monitor - transfer control to reset logic;

set time of day - modify monitor hardware clock and generate response to Management Workstation;

get time of day - read monitor hardware clock and generate response to Workstation;

set trap permit - send trap control ITM to EM and generate response to Workstation;

get trap permit - generate response to Workstation;

Control module 42 also updates parse control records within STATS when invoked by the RTP (to be described) or during overload conditions so that higher layers of parsing are dropped until the overload situation is resolved. When overload is over it restores full parsing.

Timer 30 is invoked periodically to perform general housekeeping functions. It pulses the watchdog timer at appropriate intervals. It also takes care of internal time stamping and kicking off routines like the EM routine which periodically recalculates certain numbers within the statistical database (i.e., STATS).

Real Time Parser (RTP) 32 sees all frames on the network and it determines which protocols are being used and interprets the frames. The RTP includes a protocol parser and a state machine. The protocol parser parses a received frame in the "classical" manner, layer-by-layer, lowest layer first. The parsing is performed such that the statistical objects in STATS (i.e., the network parameters for which performance data is kept) are maintained. Which layers are to have statistics stored for them is determined by a parse control record that is stored in STATS (to be described later). As each layer is parsed, the RTP invokes the appropriate functions in the statistics module (STATS) to update those statistical objects which must be changed.

The state machine within RTP 32 is responsible for tracking state as appropriate to protocols and connections. It is responsible for maintaining and updating the connection oriented statistical elements in

STATS. In order to track connection states and events, the RTP invokes a routine within the state machine. This routine determines the state of a connection based on past observed frames and keeps track of sequence numbers. It is the routine that determines if a connection is in data transfer state and if a retransmission has occurred. The objectives of the state machine are to keep a brief history of events, state transitions, and sequence numbers per connection; to detect data transfer state so that sequence tracking can begin; and to count inconsistencies but still maintain tracking while falling into an appropriate state (e.g. unknown).

RTP 32 also performs overload control by determining the number of frames awaiting processing and invoking control module 42 to update the parse control records so as to reduce the parsing depth when the number becomes too large.

Statistics Module (STATS) 36 is where Monitor 10 keeps information about the statistical objects it is charged with monitoring. A statistical object represents a network parameter for which performance information is gathered. This information is contained in an extended MIB (Management Information Base), which is updated by RTP 32 and EM 38.

STATS updates statistical objects in response to RTP invocation. There are at least four statistical object classes, namely, counters, timers, percentages (%), and meters. Each statistical object is implemented as appropriate to the object class to which it belongs. That is, each statistical object behaves such that when invoked by RTP 32 it updates and then generates an alarm if its value meets a preset threshold. (Meets means that for a high threshold the value is equal to or greater than the threshold and for a low threshold the value is

equal to or less than the threshold. Note that a single object may have both high and low thresholds.)

STATS 36 is responsible for the maintenance and initial analysis of the database. This includes coordinating access to the database variables, ensuring appropriate interlocks are applied and generating alarms when thresholds are crossed. Only STATS 36 is aware of the internal structure of the database, the rest of the system is not.

STATS 36 is also responsible for tracking events of interest in the form of various statistical reductions. Examples are counters, rate meters, and rate of change of rate meters. It initiates events based on particular statistics reaching configured limits, i.e., thresholds. The events are passed to the EM which sends a trap (i.e., an alarm) to the Management Workstation. The statistics within STATS 36 are readable from the Management Workstation on request.

STATS performs lookup on all addressing fields. It assigns new data structures to address field values not currently present. It performs any hashing for fast access to the database. More details will be presented later in this document.

Event Manager (EM) 38 extracts statistics from STATS and formats it in ways that allow the Workstation to understand it. It also examines the various statistics to see if their behavior warrants a notification to the Management Workstation. If so, it uses the SNMP Agent software to initiate such notifications.

If the Workstation asks for data, EM 38 gets the data from STATS and sends it to the Workstation. It also performs some level of analysis for statistical, accounting and alarm filtering and decides on further action (e.g. delivery to the Management Workstation).

EM 38 is also responsible for controlling the delivery of events to the Management Workstation, e.g., it performs event filtering. The action to be taken on receipt of an event (e.g. threshold exceeded in STATS) is specified by the event action associated with the threshold. The event is used as an index to select the defined action (e.g. report to Workstation, run local routine xxxx, ignore). The action can be modified by commands from the Management Workstation (e.g., turn off an alarm) or by the control module in an overload situation. An update to the event action, however, does not affect events previously processed even if they are still waiting for transmission to the Management Workstation. Discarded events are counted as such by EM 38.

EM 38 also implements a throttle mechanism to limit the rate of delivery of alarms to the console based on configured limits. This prevents the rapid generation of multiple alarms. In essence, Monitor 10 is given a maximum frequency at which alarms may be sent to the Workstation. Although alarms in excess of the maximum frequency are discarded, a count is kept of the number of alarms that were discarded.

EM 38 invokes routines from the statistics module (STATS) to perform periodic updates such as rate calculations and threshold checks. It calculates time averages, e.g., average traffic by source stations, destination stations. EM 38 requests for access to monitor control variables are passed to the control module.

EM 38 checks whether asynchronous traps (i.e., alarms) to the Workstation are permitted before generating any.

EM 38 receives database update requests from the Management Workstation and invokes the statistics module (STATS) to process these.

Message Transport Module (MTM) 34, which is DRAM based, has two distinct but closely related functions. First, it is responsible for the conversion of Workstation commands and responses from the internal format used within Monitor 10 to the format used to communicate over the network. It isolates the rest of the system from the protocol used to communicate within Management Workstation. It translates between the internal representation of data and ASN.1 used for SNMP. It performs initial decoding of Workstation requests and directs the requests to appropriate modules for processing. It implements SNMP/UDP/IP/LLC or ETHERNET protocols for LAN and SNMP/UDP/IP/SLIP protocols for serial line. It receives network management commands from the Management Workstation and delivers these to the appropriate module for action. Alarms and responses destined for the Workstation are also directed via this module.

Second, MTM 34 is responsible for the delivery and reception of data to and from the Management Workstation using the protocol appropriate to the network. Primary and backup communication paths are provided transparently to the rest of the monitor modules (e.g. LAN and dial up link). It is capable of full duplex delivery of messages between the console and monitoring module. The messages carry event, configuration, test and statistics data.

Event Timing Module (ETM) 40 keeps track of the start time and end times of user specified transactions over the network. In essence, this module monitors the responsiveness of the network at any protocol or layer specified by the user.

Address Tracking Module 42 keeps track of the node name to node address bindings on networks which implement dynamic node addressing protocols.

Memory management for Monitor 10 is handled in accordance with following guidelines. The available memory is divided into four blocks during system initialization. One block includes receive frame buffers. They are used for receiving LAN traffic and for receiving secondary link traffic. These are organized as linked lists of fixed sized buffers. A second block includes system control message blocks. They are used for intertask messages within Monitor 10 and are organized as a linked list of free blocks and multiple linked lists of in process intertask messages. A third block includes transmit buffers. They are used for creation and transmission of workstation alarms and responses and are organized as a linked list of fixed sized buffers. A fourth block is the statistics. This is allocated as a fixed size area at system initialization and managed by the statistics module during system operation.

Task Structure of Monitor;

The structure of the Monitor in terms of tasks and intertask messages is shown in Fig. 6. The rectangular blocks represent interrupt service routines, the ovals represent tasks and the circles represent input queues.

Each task in the system has a single input queue which it uses to receive all input. All inter-process communications take place via messages placed onto the input queue of the destination task. Each task waits on a (well known) input queue and processes events or inter-task messages (i.e., ITM's) as they are received. Each task returns to the kernel within an appropriate time period defined for each task (e.g. after processing a fixed number of events).

Interrupt service routines (ISR's) run on receipt of hardware generated interrupts. They invoke task level

processing by sending an ITM to the input queue of the appropriate task.

The kernel scheduler acts as the base loop of the system and calls any runnable tasks as subroutines. The determination of whether a task is runnable is made from the input queue, i.e., if this has an entry the task has work to perform. The scheduler scans the input queues for each task in a round robin fashion and invokes a task with input pending. Each task processes items from its input queue and returns to the scheduler within a defined period. The scheduler then continues the scan cycle of the input queues. This avoids any task locking out others by processing a continuously busy input queue. A task may be given an effectively higher priority by providing it with multiple entries in the scan table.

Database accesses are generally performed using access routines. This hides the internal structure of the database from other modules and also ensures that appropriate interlocks are applied to shared data.

The EM processes a single event from the input queue and then returns to the scheduler.

The MTM Xmit task processes a single event from its input queue and then returns control to the scheduler. The MTM Recv task processes events from the input queue until it is empty or a defined number (e.g. 10) events have been processed and then returns control to the scheduler.

The timer task processes a single event from the input queue and then returns control to the scheduler.

RTP continues to process frames until the input queue is empty or it has processed a defined number (e.g. 10) frames. It then returns to the scheduler.

The following sections contain a more detailed description of some of the above-identified software modules.

The Statistics Module (STATS):

The functions of the statistics module are:

* to define statistics records;
* to allocate and initialize statistics records;
* to provide routines to lookup statistics records, e.g. lookup_id_addr;
* to provide routines to manipulate the statistics within the records, e.g. stats_age, stats_incr and stats_rate;
* to provide routines to free statistics records, e.g. stats_allocate and stats_deallocate

It provides these services to the Real Time Parser (RTP) module and to the Event Manager (EM) module.

STATS defines the database and it contains subroutines for updating the statistics which it keeps.

STATS contains the type definitions for all statistics records (e.g. DLL, IP, TCP statistics). It provides an initialization routine whose major function is to allocate statistics records at startup from cacheable memory. It provides lookup routines in order to get at the statistics. Each type of statistics record has its own lookup routine (e.g. lookup_ip_address) which returns a pointer to a statistics record of the appropriate type or NULL.

As a received frame is being parsed, statistics within statistics records need to be manipulated (e.g. incremented) to record relevant information about the frame. STATS provides the routines to manipulate those statistics. For example, there is a routine to update counters. After the counter is incremented/decremented and if there is a non-zero threshold associated with the counter, the internal routine compares its value to the threshold. If the threshold has been exceeded, the Event Manager is signaled in order to send a trap to the Workstation. Besides manipulating statistics, these

routines, if necessary, signal the Event Manager via an Intertask Message (ITM) to send a trap to the Management Workstation.

The following is an example of some of the statistics records that are kept in STATS.

- o monitor statistics
- o mac statistics for segment
- o llc statisics for segment
- o statistics per ethernet/lsap type for segment
- o ip statistics for segment
- o icmp statistics for segment
- o tcp statistics for segment
- o udp statistics for segment
- o nfs statistics for segment
- o ftp control statistics for segment
- o ftp data statistics for segment
- o telnet statistics for segment
- o smtp statistics for segment
- o arp statistics for segment

- o statistics per mac address
- o statistics per ethernet type/lasp per mac address
- o statistics per ip address (includes icmp)
- o statistics per tcp socket
- o statistics per udp socket
- o statistics per nfs socket
- o statistics per ftp control socket
- o statistics per ftp data socket
- o statistics per telnet socket
- o statistics per smtp socket
- o arp statistics per ip address

- o statistics per mac address pair
- o statistics per ip pair (includes icmp)

- o statistics per tcp connection
- o statistics per udp pair
- o statistics per nfs pair
- o statistics per ftp control connection
- o statistics per ftp data connection
- o statistics per telnet connection
- o statistics per smtp connection

- o connection histories per udp and tcp socket

All statistics are organized similarly across protocol types. The details of the data structures for the DLL level are presented later.

As noted earlier, there are four statistical object classes (i.e., variables), namely, counts, rates, percentages (%), and meters. They are defined and implemented as follows.

A count is a continuously incrementing variable which rolls around to 0 on overflow. It may be reset on command from the user (or from software). A threshold may be applied to the count and will cause an alarm when the threshold count is reached. The threshold count fires each time the counter increments past the threshold value. For example, if the threshold is set to 5, alarms are generated when the count is 5, 10, 15,...

A rate is essentially a first derivative of a count variable. The rate is calculated at a period appropriate to the variable. For each rate variable, a minimum, maximum and average value is maintained. Thresholds may be set on high values of the rate. The maximums and minimums may be reset on command. The threshold event is triggered each time the rate calculated is in the threshold region.

As commonly used, the % is calculated at a period appropriate to the variable. For each % variable a

minimum, maximum and average value is maintained. A threshold may be set on high values of the %. The threshold event is triggered each time the % calculated is in the threshold region.

Finally, a meter is a variable which may take any discrete value within a defined range. The current value has no correlation to past or future values. A threshold may be set on a maximum and/or minimum value for a meter.

The rate and % fields of network event variables are updated differently than counter or meter fields in that they are calculated at fixed intervals rather than on receipt of data from the network.

Structures for statistics kept on a per address or per address pair basis are allocated at initialization time. There are several sizes for these structures. Structures of the same size are linked together in a free pool. As a new structure is needed, it is obtained from a free queue, initialized, and linked into an active list. Active lists are kept on a per statistics type basis.

As an address or address pair (e.g. mac, ip, tcp...) is seen, RTP code calls an appropriate lookup routine. The lookup routine scans active statistics structures to see if a structure has already been allocated for the statistics. Hashing algorithms are used in order to provide for efficient lookup. If no structure has been allocated, the lookup routine examines the appropriate parse control records to determine whether statistics should be kept, and, if so, it allocates a structure of the appropriate size, initializes it and links it into an active list.

Either the address of a structure or a NULL is returned by these routines. If NULL is returned, the RTP does not stop parsing, but it will not be allowed to

store the statistics for which the structure was requested.

The RTP updates statistics within the data base as it runs. This is done via macros defined for the RTP. The macros call on internal routines which know how to manipulate the relevant statistic. If the pointer to the statistics structure is NULL, the internal routine will not be invoked.

The EM causes rates to be calculated. The STATS module supplies routines (e.g. stats_rate) which must be called by the EM in order to perform the rate calculations. It also calls subroutines to reformat the data in the database in order to present it to the Workstation (i.e., in response to a get from the Workstation).

The calculation algorithms for the rate and % fields of network event variables are as follows.

The following rates are calculated in units per second, at the indicated (approximate) intervals:

1. 10 second intervals:
   e.g. DLL frame, byte, ethernet, 802.3, broadcast, multicast rates
2. 60 second intervals
   e.g., all DLL error, ethertype/dsap rates
   all IP rates.
   TCP packets, bytes, errors, retransmitted packets, retransmitted bytes, acks, rsts
   UDP packet, error, byte rates
   FTP file transfer, byte transfer, error rates

For these rates, the new average replaces the previous value directly. Maximum and minimum values are retained until reset by the user.

The following rates are calculated in units per hour at the indicated time intervals:

1. 15 minute interval.

e.g., TCP - connection rate
Telnet connection rate
FTP session rate

The hourly rate is calculated from a sum of the last twelve 5 minute readings, as obtained from the buckets for the pertinent parameter. Each new reading replaces the oldest of the twelve values maintained. Maximum and minimum values are retained until reset by the user.

There are a number of other internal routines in STATS. For example, all statistical data collected by the Monitor is subject to age out. Thus, if no activity is seen for an address (or address pair) in the time period defined for age out, then the data is discarded and the space reclaimed so that it may be recycled. In this manner, the Monitor is able to use the memory for active elements rather than stale data. The user can select the age out times for the different components. The EM periodically kicks off the aging mechanism to perform this recycling of resources. STATS provides the routines which the EM calls, e.g. stats_age.

There are also routines in STATS to allocate and de-allocate Statistics, e.g., stats_allocate and stats_de-allocate. The allocate routine is called when stations and dialogs are picked up by the Network Monitor. The de-allocate routine is called by the aging routines when a structure is to be recycled.

<u>The Data Structures in STATS</u>

The general structure of the database within STATS is illustrated by Figs. 7a-c, which shows information that is maintained for the Data Link Layer (DLL) and its organization. A set of data structures is kept for each address associated with the layer. In this case there are three relevant addresses, namely a segment address, indicating which segment the node is on, a MAC address

for the node on the segment, and an address which identifies the dialog occurring over that layer. The dialog address is the combination of the MAC addresses for the two nodes which make up the dialog. Thus, the overall data structure has three identifiable components: a segment address data structure (see Fig. 7a), a MAC address data structure (see Fig. 7b) and a dialog data structure (see Fig. 7c).

The segment address structure includes a doubly linked list 102 of segment address records 104, each one for a different segment address. Each segment address record 104 contains a forward and backward link (field 106) for forward and backward pointers to neighboring records and a hash link (field 108). In other words, the segment address records are accessed by either walking down the doubly linked list or by using a hashing mechanism to generate a pointer into the doubly linked list to the first record of a smaller hash linked list. Each record also contains the address of the segment (field 110) and a set of fields for other information. Among these are a flags field 112, a type field 114, a parse_control field 116, and an EM_control field 118. Flags field 112 contains a bit which indicates whether the identified address corresponds to the address of another Network Monitor. This field only has meaning in the MAC address record and not in the segment or dialog address record. Type field 114 identifies the MIB group which applies to this address. Parse control field 116 is a bit mask which indicates what subgroups of statistics from the identified MIB group are maintained, if any. Flags field 112, type field 114 and parse control field 116 make up what is referred to as the parse control record for this MAC address. The Network Monitor uses a default value for parse control field 116 upon initialization or whenever a new node is detected.

The default value turns off all statistics gathering. The statistics gathering for any particular address may subsequently be turned on by the Workstation through a Network Monitor control command that sets the appropriate bits of the parse control field to one.

EM_control field 118 identifies the subgroups of statistics within the MIB group that have changed since the EM last serviced the database to update rates and other variables. This field is used by the EM to identify those parts of STATS which must be updated or for which recalculations must be performed when the EM next services STAT.

Each segment address record 104 also contains three fields for time related information. There is a start_time field 120 for the time that is used to perform some of the rate calculations for the underlying statistics; a first_seen field 122 for the time at which the Network Monitor first saw the communication; and a last_seen field 124 for the time at which the last communication was seen. The last_seen time is used to age out the data structure if no activity is seen on the segment after a preselected period of time elapses. The first_seen time is a statistic which may be of interest to the network manager and is thus retrievable by the Management Workstation for display.

Finally, each segment address record includes a stats_pointer field 126 for a pointer to a DLL segment statistics data structure 130 which contains all of the statistics that are maintained for the segment address. If the bits in parse_control field 116 are all set to off, indicating that no statistics are to be maintained for the address, then the pointer in stats_pointer field 126 is a null pointer.

The list of events shown in data structure 130 of Fig. 7a illustrates the type of data that is collected

for this address when the parse control field bits are set to on. Some of the entries in DLL segment statistics data structure 130 are pointers to buckets for historical data. In the case where buckets are maintained, there are twelve buckets each of which represents a time period of five minutes duration and each of which generally contains two items of information, namely, a count for the corresponding five minute time period and a MAX rate for that time period. MAX rate records any spikes which have occurred during the period and which the user may not have observed because he was not viewing that particular statistic at the time.

At the end of DLL segment statistics data structure 130, there is a protocol_Q pointer 132 to a linked list 134 of protocol statistics records 136 identifying all of the protocols which have been detected running on top of the DLL layer for the segment. Each record 136 includes a link 138 to the next record in the list, the identity of the protocol (field 140), a frames count for the number of frames detected for the identified protocol (field 142); and a frame rate (field 144).

The MAC address data structure is organized in a similar manner to that of the segment data structure (see Fig. 7b). There is a doubly linked list 146 of MAC address records 148, each of which contains the same type of information as is stored in DLL segment address records 104. A pointer 150 at the end of each MAC address record 148 points to a DLL address statistics data structure 152, which like the DLL segment address data structure 130, contains fields for all of the statitics that are gathered for that DLL MAC address. Examples of the particular statistics are shown in Fig. 7b.

At the end of DLL address statistics data structure 152, there are two pointer fields 152 and 154, one for a pointer to a record 158 in a dialog link queue 160, and the other for a pointer to a linked list 162 of protocol statistics records 164. Each dialog link queue entry 158 contains a pointer to the next entry (field 168) in the queue and it contains a dialog_addr pointer 170 which points to an entry in the DLL dialog queue which involves the MAC address. (see Fig. 7c). Protocol statistics records 164 have the same structure and contain the same categories of information as their counterparts hanging off of DLL segment statistics data structure 130.

The above-described design is repeated in the DLL dialog data structures. That is, dialog record 172 includes the same categories of information as its counterpart in the DLL segment address data structure and the MAC address data structure. The address field 174 contains the addresses of both ends of the dialog concatenated together to form a single address. The first and second addresses within the single address are arbitrarily designated nodes 1 and 2, respectively. In the stats_pointer field 176 there is a pointer to a dialog statistics data structure 178 containing the relevant statistics for the dialog. The entries in the first two fields in this data structure (i.e., fields 180 and 182) are designated protocol entries and protocols. Protocol entries is the number of different protocols which have been seen between the two MAC addresses. The protocols that have been seen are enumerated in the protocols field 182.

DLL dialog statistics data structure 178, illustrated by Fig. 7c, includes several additional fields of information which only appear in these structures for dialogs for which state information can be

kept (e.g. TCP connection). The additional fields identify the transport protocol (e.g., TCP) (field 184) and the application which is running on top of that protocol (field 186). They also include the identity of the initiator of the connection (field 188), the state of the connection (field 190) and the reason that the connection was closed, when it is closed (field 192). Finally, they also include a state_pointer (field 194) which points to a history data structure that will be described in greater detail later. Suffice it to say, that the history data structure contains a short history of events and states for each end of the dialog. The state machine uses the information contained in the history data structure to loosely determine what the state of each of the end nodes is throughout the course of the connection. The qualifier "loosely" is used because the state machine does not closely shadow the state of the connection and thus is capable of recovering from loss of state due to lost packets or missed communications.

The above-described structures and organization are used for all layers and all protocols within STATS.

<u>Real Time Parser (RTP)</u>

The RTP runs as an application task. It is scheduled by the Real Time Kernel scheduler when received frames are detected. The RTP parses the frames and causes statistics, state tracking, and tracing operations to be performed.

The functions of the RTP are:

* obtain frames from the RTP Input Queue;
* parse the frames;
* maintain statistics using routines supplied by the STATS module;
* maintain protocol state information;

* notify the MTM via an ITM if a frame has been received with the Network Monitor's address as the destination address; and
* notify the EM via an ITM if a frame has been received with any Network Monitor's address as the source address.

The design of the RTP is straightforward. It is a collection of routines which perform protocol parsing. The RTP interfaces to the Real Time Kernel in order to perform RTP initialization, to be scheduled in order to parse frames, to free frames, to obtain and send an ITM to another task; and to report fatal errors. The RTP is invoked by the scheduler when there is at least one frame to parse. The appropriate parse routines are executed per frame. Each parse routine invokes the next level parse routine or decides that parsing is done. Termination of the parse occurs on an error or when the frame has been completely parsed.

Each parse routine is a separately compilable module. In general, parse routines share very little data. Each knows where to begin parsing in the frame and the length of the data remaining in the frame.

The following is a list of the parse routines that are available within RTP for parsing the different protocols at the various layers.

Data Link Layer Parse - rtp_dll_parse:

This routine handles Ethernet, IEEE 802.3, IEEE 802.2, and SNAP. See RFC 1010, Assigned Numbers for a description of SNAP (Subnetwork Access Protocol).

Address Resolution Protocol Parse - rtp_arp_parse

ARP is parsed as specified in RFC 826.

Internet Protocol Parse - rtp_ip_parse

IP Version 4 is parsed as specified in RFC 791 as amended by RFC 950, RFC 919, and RFC 922.

Internet Control Message Protocol Parse - rtp_icmp_parse

ICMP is parsed as specified in RFC 792.

Unit Data Protocol Parse - rtp_udp_parse

UDP is parsed as specified in RFC 768.

Transmission Control Protocol Parse - rtp_tcp_parse

TCP is parsed as specified in RFC 793.

Simple Mail Transfer Protocol Parse - rtp_smtp_parse

SMTP is parsed as specified in RFC 821.

File Transfer Protocol Parse - rtp_ftp_parse

FTP is parsed as specified in RFC 959.

Telnet Protocol Parse - rtp_telnet_parse

The Telnet protocol is parsed as specified in RFC 854.

Network File System Protocol Parse - rpt_nfs_parse

The NFS protocol is parsed as specified in RFC 1094.

The RTP calls routines supplied by STATS to look up data structures. By calling these lookup routines, global pointers to data structures are set up. Following are examples of the pointers to statistics data structures that are set up when parse routines call Statistics module lookup routines.

mac_segment, mac_dst_segment, mac_this_segment,
mac_src, mac_dst, mac_dialog
ip_src_segment, ip_dst_segment, ip_this_segment,
ip_src, ip_dst, ip_dialog
tcp_src_segment, tcp_dst_segment,
tcp_this_segment,
tcp_src, tcp_dst, tcp_src_socket, tcp_dst_socket,
tcp_connection

The mac_src and mac_dst routines return pointers to the data structures within STATS for the source MAC address and the destination MAC address, respectively. The lookup_mac_dialog routine returns a pointer to the data structure within STATS for the dialog between the

two nodes on the MAC layer. The other STATS routines supply similar pointers for data structures relevant to other protocols.

The RTP routines are aware of the names of the statistics that must be manipulated within the data base (e.g. frames, bytes) but are not aware of the structure of the data. When a statistic is to be manipulated, the RTP routine invokes a macro which manipulates the appropriate statistics in data structures. The macros use the global pointers which were set up during the lookup process described above.

After a frame has been parsed (whether the parse was successful or not), the RTP routine examines the destination mac and ip addresses. If either of the addresses is that of the Network Monitor, RTP obtains a low priority ITM, initializes it, and sends the ITM to the MTM task. One of the fields of the ITM contains the address of the buffer containing the frame.

The RTP must hand some received frames to the EM in order to accomplish the autotopology function (described later). After a frame has been parsed (whether the parse was successful or not), the RTP routine examines the source mac and ip addresses. If either of the addresses is that of another Network Monitor, RTP obtains a low priority ITM, initializes it and sends the ITM to the EM task. The address data structure (in particular, the flags field of the parse control record) within STATS for the MAC or the IP address indicates whether the source address is that of another Network Monitor. One of the fields of the ITM contains the address of the buffer containing the frame.

The RTP receives traffic frames from the network for analysis. RTP operation may be modified by sending control messages to the Monitor. RTP first parses these messages, then detects that the messages are destined for

the Monitor and passes them to the MTM task. Parameters which affect RTP operation may be changed by such control messages.

The general operation of the RTP upon receipt of a traffic frame is as follows:

```
Get next frame from input queue
get address records for these stations
For each level of active parsing
{
get pointer to start of protocol header
call layer parse routine
determine protocol at next level
set pointer to start of next layer protocol

}end of frame parsing
if this is a monitor command add to MTM input
queue
if this frame is from another monitor, pass
to EM
check for overload -if yes tell control
```

__The State Machine__:

In the described embodiment, the state machine determines and keeps state for both addresses of all TCP connections. TCP is a connection oriented transport protocol, and TCP clearly defines the connection in terms of states of the connection. There are other protocols which do not explicitly define the communication in terms of state, e.g. connectionless protocols such as NFS. Nevertheless, even in the connectionless protocols there is implicitly the concept of state because there is an expected order to the events which will occur during the course of the communication. That is, at the very least, one can identify a beginning and an end of the communication, and usually some sequence of events which will occur during the course of the communication. Thus,

even though the described embodiment involves a connection oriented protocol, the principles are applicable to many connectionless protocols or for that matter any protocol for which one can identify a beginning and an end to the communication under that protocol.

Whenever a TCP packet is detected, the RTP parses the information for that layer to identify the event associated with that packet. It then passes the identified event along with the dialog identifier to the state machine. For each address of the two parties to the communication, the state machine determines what the current state of the node is. The code within the state machine determines the state of a connection based upon a set of rules that are illustrated by the event/state table shown in Fig. 8.

The interpretation of the event/state table is as follows. The top row of the table identifies the six possible states of a TCP connection. These states are not the states defined in the TCP protocol specification. The left most column identifies the eight events which may occur during the course of a connection. Within the table is an array of boxes, each of which sits at the intersection of a particular event/state combination. Each box specifies the actions taken by the state machine if the identified event occurs while the connection is in the identified state. When the state machine receives a new event, it may perform three types of action. It may change the recorded state for the node. The state to which the node is changed is specified by the S="STATE" entry located at the top of the box. It may increment or decrement the appropriate counters to record the information relevant to that event's occurrence. (In the table, incrementing and decrementing are signified by the ++ and the -- symbols, respectively, located after the

identity of the variable being updated.) Or the state machine may take other actions such as those specified in the table as start close timer, Look_for_Data_State, or Look_at_History (to be described shortly). The particular actions which the state machine takes are specified in each box. An empty box indicates that no action is taken for that particular event/state combination. Note, however, that the occurrence of an event is also likely to have caused the update of statistics within STATS, if not by the state machine, then by some other part of the RTP. Also note that it may be desirable to have the state machine record other events, in which case the state table would be modified to identify those other actions.

Two events appearing on the table deserve further explanation, namely, close timer expires and inactivity timer expires. The close timer, which is specified by TCP, is started at the end of a connection and it establishes a period during which any old packets for the connection which are received are thrown away (i.e., ignored). The inactivity timer is not specified by TCP but rather is part of the Network Monitor's resource management functions. Since keeping statistics for dialogs (especially old dialogs) consumes resources, it is desirable to recycle resources for a dialog if no activity has been seen for some period of time. The inactivity timer provides the mechanism for accomplishing this. It is restarted each time an event for the connection is received. If the inactivity timer expires (i.e., if no event is received before the timer period ends), the connection is assumed to have gone inactive and all of the resources associated with the dialog are recycled. This involves freeing them up for use by other dialogs.

The other states and events within the table differ from but are consistent with the definitions provided by TCP and should be self evident in view of that protocol specification.

The event/state table can be read as follows. Assume, for example, that node 1 is in DATA state and the RTP receives another packet from node 1 which it determines to be a TCP FIN packet. According to the entry in the table at the intersection of FIN/DATA (i.e., event/state), the state machine sets the state of the connection for node 1 to CLOSING, it decrements the active connections counter and it starts the close timer. When the close timer expires, assuming no other events over that connection have occurred, the state machine sets node 1's state to CLOSED and it starts the inactivity timer. If the RTP sends another SYN packet to reinitiate a new connection before the inactive timer expires, the state machine sets node 1's state to CONNECTING (see the SYN/CLOSED entry) and it increments an after close counter.

When a connection is first seen, the Network Monitor sets the state of both ends of the connection to UNKNOWN state. If some number of data and acknowledgment frames are seen from both connection ends, the states of the connection ends may be promoted to DATA state. The connection history is searched to make this determination as will be described shortly.

Referring to Figs. 9a-b, within STATS there is a history data structure 200 which the state machine uses to remember the current state of the connection, the state of each of the nodes participating in the connection and a short history of state related information. History data structure 200 is identified by a state_pointer found at the end of the associated dialog statistics data structure in STATS (see Fig. 7c). Within

history data structure 200, the state machine records the current state of node 1 (field 202), the current state of node 2 (field 206) and other data relating to the corresponding node (fields 204 and 208). The other data includes, for example, the window size for the receive and transmit communications, the last detected sequence numbers for the data and acknowledgment frames, and other data transfer information.

History data structure 200 also includes a history table (field 212) for storing a short history of events which have occurred over the connection and it includes an index to the next entry within the history table for storing the information about the next received event (field 210). The history table is implemented as a circular buffer which includes sufficient memory to store, for example, 16 records. Each record, shown in Fig. 9b, stores the state of the node when the event was detected (field 218), the event which was detected (i.e., received) (field 220), the data field length (field 222), the sequence number (field 224), the acknowledgment sequence number (field 226) and the identity of the initiator of the event, i.e., either node 1 or node 2 or 0 if neither (field 228).

Though the Network Monitor operates in a promiscuous mode, it may occasionally fail to detect or it may, due to overload, lose a packet within a communication. If this occurs the state machine may not be able to accurately determine the state of the connection upon receipt of the next event. The problem is evidenced by the fact that the next event is not what was expected. When this occurs, the state machine tries to recover state by relying on state history information stored in the history table in field 212 to deduce what the state is. To deduce the current state from historical information, the state machine uses one of the

two previously mentioned routines, namely, Look_for_Data_State and Look_at_History.

Referring to Fig. 10, Look_for_Data_State routine 230 searches back through the history one record at a time until it finds evidence that the current state is DATA state or until it reaches the end of the circular buffer (step 232). Routine 230 detects the existence of DATA state by determining whether node 1 and node 2 each have had at least two data events or two acknowledgment combinations with no intervening connect, disconnect or abort events (step 234). If such a sequence of events is found within the history, routine 230 enters both node 1 and node 2 into DATA state (step 236), it increments the active connections counter (step 238) and then it calls a Look_for_Initiator routine to look for the initiator of the connection (step 240). If such a pattern of events is not found within the history, routine 230 returns without changing the state for the node (step 242).

As shown in Fig. 11, Look_for_Initiator routine 240 also searches back through the history to detect a telltale event pattern which identifies the actual initiator of the connection (step 244). More specifically, routine 240 determines whether nodes 1 and 2 each sent connect-related packets. If they did, routine 240 identifies the initiator as the first node to send a connect-related packet (step 246). If the search is not successful, the identity of the connection initiator remains unknown (step 248).

The Look_at_History routine is called to check back through the history to determine whether data transmissions have been repeated. In the case of retransmissions, the routine calls a Look_for_Retransmission routine 250, the operation of which is shown in Fig. 12. Routine 250 searches back through the history (step 252) and checks whether the

same initiator node has sent data twice (step 254). It detects this by comparing the current sequence number of the packet as provided by the RTP with the sequence numbers of data packets that were previously sent as reported in the history table. If a retransmission is spotted, the retransmission counter in the dialog statistics data structure of STATS is incremented (step 256). If the sequence number is not found within the history table, indicating that the received packet does not represent a retransmission, the retransmission counter is not incremented (step 258).

Other statistics such as Window probes and keep alives may also be detected by looking at the received frame, data transfer variables, and, if necessary, the history.

Even if frames are missed by the Network Monitor, because it is not directly "shadowing" the connection, the Network Monitor still keeps useful statistics about the connection. If inconsistencies are detected the Network Monitor counts them and, where appropriate, drops back to UNKNOWN state. Then, the Network Monitor waits for the connection to stabilize or deteriorate so that it can again determine the appropriate state based upon the history table.

<u>Principal Transactions of Network Monitor Modules</u>:

The transactions which represent the major portion of the processing load within the Monitor, include monitoring, actions on threshold alarms, processing database get/set requests from the Management Workstation, and processing monitor control requests from the Management Workstation. Each of these mechanisms will now be briefly described.

Monitoring involves the message sequence shown in Fig. 13. In that figure, as in the other figures involving message sequences, the numbers under the

heading SEQ. identify the major steps in the sequence. The following steps occur:

1. ISR puts Received traffic frame ITM on RTP input queue
2. request address of pertinent data structure from STATS (get parse control record for this station)
3. pass pointer to RTP
4. update statistical objects by call to statistical update routine in STATS using pointer to pertinent data structure
5. parse completed - release buffers

The major steps which follow a statistics threshold event (i.e., an alarm event) are shown in Fig. 14. The steps are as follows:

1. statistical object update causes threshold alarm
2. STATS generates threshold event ITM to event manager (EM)
3. look up appropriate action for this event
4. perform local event processing
5. generate network alarm ITM to MTM Xmit (if required)
6. format network alarm trap for Workstation from event manager data
7. send alarm to Workstation

The major steps in processing of a database update request (i.e., a get/set request) from the Management Workstation are shown in Fig. 15. The steps are as follows:

1. LAN ISR receives frame from network and passes it to RTP for parsing
2. RTP parses frame as for any other traffic on segment.
3. RTP detects frame is for monitor and sends received Workstation message over LAN ITM to MTM Recv.

4. MTM Recv processes protocol stack.
5. MTM Recv sends database update request ITM to EM.
6. EM calls STATS to do database read or database write with appropriate IMPB
7. STATS performs database access and returns response to EM.
8. EM encodes response to Workstation and sends database update response ITM to MTM Xmit
9. MTM Xmit transmits.

The major steps in processing of a monitor control request from the Management Workstation are shown in Fig. 16. The steps are as follows:

1. Lan ISR receives frame from network and passes received frame ITM to RTP for parsing.
2. RTP parses frame as for any other traffic on segment.
3. RTP detects frame is for monitor and sends received workstation message over LAN ITM to MTM Recv.
4. MTM Recv processes protocol stack and decodes workstation command.
5. MTM Recv sends request ITM to EM.
6. EM calls Control with monitor control IMPB.
7. Control performs requested operation and generates response to EM.
8. EM sends database update response ITM to MTM Xmit.
9. MTM Xmit encodes response to Workstation and transmits.

<u>The Monitor/Workstation Interface</u>:

The interface between the Monitor and the Management Workstation is based on the SNMP definition (RFC 1089 SNMP; RFC 1065 SMI; RFC 1066 SNMP MIB - Note: RFC means Request for Comments). All five SNMP PDU types are supported:

get-request

get-next-request
get-response
set-request
trap

The SNMP MIB extensions are designed such that where possible a user request for data maps to a single complex MIB object. In this manner, the get-request is simple and concise to create, and the response should contain all the data necessary to build the screen. Thus, if the user requests the IP statistics for a segment this maps to an IP Segment Group.

The data in the Monitor is keyed by addresses (MAC, IP) and port numbers (telnet, FTP). The user may wish to relate his data to physical nodes entered into the network map. The mapping of addresses to physical nodes is controlled by the user (with support from the Management Workstation system where possible) and the Workstation retains this information so that when a user requests data for node 'Joe' the Workstation asks the Monitor for the data for the appropriate address(es). The node to address mapping need not be one to one.

Loading and dumping of monitors uses TFTP (Trivial File Transfer Protocol). This operates over UDP as does SNMP. The Monitor to Workstation interface follows the SNMP philosophy of operating primarily in a polled mode. The Workstation acts as the master and polls the Monitor slaves for data on a regular (configurable) basis.

The information communicated by the SNMP is represented according to that subset of ASN.1 (ISO 8824 Specification of ASN.1) defined in the Internet standard Structure of Management Information (SMI - RFC 1065). The subset of the standard Management Information Base (MIB) (RFC 1066 SNMP MIB) which is supported by the Workstation is defined in Appendix III. The added value provided by the Workstation is encoded as enterprise

specific extensions to the MIB as defined in Appendix IV. The format for these extensions follows the SMI recomendations for object identifiers so that the Workstation extensions fall in the subtree 1.3.6.1.4.1.x.1. where x is an enterprise specific node identifier assigned by the IAB.

Appendix V is a summary of the network variables for which data is collected by the Monitor for the extended MIB and which can be retrieved by the Workstation. The summary includes short decriptions of the meaning and significance of the variables, where appropriate.

<u>The Management Workstation</u>:

The Management Workstation is a SUN Sparcstation (also referred to as a Sun) available from Sun Microsystems, Inc. It is running the Sun flavor of Unix and uses the Open Look Graphical User Interface (GUI) and the SunNet Manager as the base system. The options required are those to run SunNet Manager with some additional disk storage requirement.

The network is represented by a logical map illustrating the network components and the relationships between them, as shown in Fig. 17. A hierarchical network map is supported with navigation through the layers of the hierarchy, as provided by SNM. The Management Workstation determines the topology of the network and informs the user of the network objects and their connectivity so that he can create a network map. To assist with the map creation process, the Management Workstation attempts to determine the stations connected to each LAN segment to which a Monitor is attached. Automatic determination of segment topology by detecting stations is performed using the autotopology algorithms as described in copending U.S. Patent Application S.N. ***,*** entitled "Automatic Topology Monitor for Multi-

Segment Local Area Network" filed on January 14, 1991 (Attorney Docket No. 13283-NE.APP), incorporated herein by reference.

In normal operation, each station in the network is monitored by a single Monitor that is located on its local segment. The initial determination of the Monitor responsible for a station is based on the results of the autotopology mechanism. The user may override this initial default if required.

The user is informed of new stations appearing on any segment in the network via the alarm mechanism. As for other alarms, the user may select whether stations appearing on and disappearing from the network segment generate alarms and may modify the times used in the aging algorithms. When a new node alarm occurs, the user must add the new alarm to the map using the SNM tools. In this manner, the SNM system becomes aware of the nodes.

The sequence of events following the detection of a new node is:

1. the location of the node is determined automatically for the user.
2. the Monitor generates an alarm for the user indicating the new node and providing some or all of the following information:
    - mac address of node
    - ip address of node
    - segment that the node is believed to be located on
    - Monitor to be responsible for the node
3. the user must select the segment and add the node manually using the SNM editor

4. The update to the SNM database will be detected and the file reread. The Workstation database is reconstructed and the parse control records for the Monitors updated if required.
5. The Monitor responsible for the new node has its parse control record updated via SNMP set request(s).

An internal record of new nodes is required for the autotopology. When a new node is reported by a Network Monitor, the Management Workstation needs to have the previous location information in order to know which Network Monitors to involve in autotopology. For example, two nodes with the same IP address may exist in separate segments of the network. The history makes possible the correlation of the addresses and it makes possible duplicate address detection.

Before a new Monitor can communicate with the Management Workstation via SNMP it needs to be added to the SNM system files. As the SNM files are cached in the database, the file must be updated and the SNM system forced to reread it.

Thus, on the detection of a new Monitor the following events need to occur in order to add the Monitor to the Workstation:

1. The Monitor issues a trap to the Management Workstation software and requests code to be loaded from the Sun Microsystems boot/load server.
2. The code load fails as the Monitor is not known to the unix networking software at this time.
3. The Workstation confirms that the new Monitor does not exceed the configured system limits (e.g. 5 Monitors per

Workstation) and terminates the initialization sequence if limits are exceeded. An alarm is issued to the user indicating the presence of the new Monitor and whether it can be supported.

4. The user adds the Monitor to the SNMP.HOSTS file of the SNM system, to the etc/hosts file of the Unix networking system and to the SNM map.
5. When the files have been updated the user resets the Monitor using the set tool (described later).
6. The Monitor again issues a trap to the Management Workstation software and requests code to be loaded from the Sun boot/load server.
7. The code load takes place and the Monitor issues a trap requesting data from the Management Workstation.
8. The Monitor data is issued using SNMP set requests.

Note that on receiving the set request, the SNMP proxy rereads in the (updated) SNMP.HOSTS file which now includes the new Monitor. Also note that the SNMP hosts file need only contain the Monitors, not the entire list of nodes in the system.

9. On completion of the set request(s) the Monitor run command is issued by the Workstation to bring the Monitor on line.

The user is responsible for entering data into the SNM database manually. During operation, the Workstation monitors the file write date for the SNM database. When this is different from the last date read, the SNM database is reread and the Workstation database reconstructed. In this manner, user updates to the SNM

database are incorporated into the Workstation database as quickly as possible without need for the user to take any action.

When the Workstation is loaded, the database is created from the data in the SNM file system (which the user has possibly updated). This data is checked for consistency and for conformance to the limits imposed by the Workstation at this time and a warning is generated to the user if any problems are seen. If the data errors are minor the system continues operation; if they are fatal the user is asked to correct them and Workstation operation terminates.

The monitoring functions of the Management Workstation are provided as an extension to the SNM system. They consist of additional display tools (i.e., summary tool, values tool, and set tool) which the user invokes to access the Monitor options and a Workstation event log in which all alarms are recorded.

As a result of the monitoring process, the Monitor makes a large number of statistics available to the operator. These are available for examination via the Workstation tools that are provided. In addition, the Monitor statistics (or a selected subset thereof) can be made visible to any SNMP manager by providing it with knowledge of the extended MIB. A description of the statistics maintained are described elswhere.

Network event statistics are maintained on a per network, per segment and per node basis. Within a node, statistics are maintained on a per address (as appropriate to the protocol layer - IP address, port number, ...) and per connection basis. Per network statistics are always derived by the Workstation from the per segment variables maintained by the Monitors. Subsets of the basic statistics are maintained on a node to node and segment to segment basis.

If the user requests displays of segment to segment traffic, the Workstation calculates this data as follows. The inter segment traffic is derived from the node to node statistics for the intersecting set of nodes. Thus, if segment A has nodes 1, 2, and 3 and segment B has nodes 20, 21, and 22, then summing the node to node traffic for

1 -> 20,21,22
2 -> 20,21,22
3 -> 20,21,22

produces the required result. On-LAN/off-LAN traffic for segments is calculated by a simply summing node to node traffic for all stations on the LAN and then subtracting this from total segment counts.

Alarms are reported to the user in the following ways:

1. Alarms received are logged in a Workstation log.
2. The node which the alarm relates to is highlighted on the map.
3. The node status change is propagated up through the (map) hierarchy to support the case where the node is not visible on the screen. This is as provided by SNM.

Summary Tool

After the user has selected an object from the map and invokes the display tools, the summary tool generates the user's initial screen at the Management Workstation. It presents a set of statistical data selected to give an overview of the operational status of the object (e.g., a selected node or segment). The Workstation polls the Monitor for the data required by the Summary Tool display screens.

The Summary Tool displays a basic summary tool screen such as is shown in Fig. 18. The summary tool screen has three panels, namely, a control panel 602, a

values panel 604, and a dialogs panel 606. The control panel includes the indicated mouse activated bottons. The functions of each of the buttons is as follows. The file button invokes a traditional file menu. The view button invokes a view menu which allows the user to modify or tailor the visual protperties of the tool. The properties button invokes a properties menu containing choices for viewing and sometimes modifying the properties of objects. The tools button invokes a tools menu which provides access to the other Workstation tools, e.g. Values Tool.

The Update Interval field allows the user to specify the frequency at which the displayed statistics are updated by polling the Monitor. The Update Once button enables the user to retrieve a single screen update. When the Update Once button is invoked not only is the screen updated but the update interval is automatically set to "none".

The type field enables the user to specify the type of network objects on which to operate, i.e., segment or node.

The name button invokes a pop up menu containing an alphabetical list of all network objects of the type selected and apply and reset buttons. The required name can then be selected from the (scrolling) list and it will be entered in the name field of the summary tool when the apply button is invoked. Alternatively, the user may enter the name directly in the summary tool name field.

The protocol button invokes a pop up menu which provides an exclusive set of protocol layers which the user may select. Selection of a layer copies the layer name into the displayed field of the summary tool when the apply operation is invoked. An example of a protocol selection menu is shown in Fig. 19. It displays the

available protocols in the form of a protocol tree with multiple protocol familes. The protocol selection is two dimensional. That is, the user first selects the protocol family and then the particular layer within that family.

As indicated by the protocol trees shown in Fig. 19, the capabilities of the Monitor can be readily extended to handle other protocol families. The particular ones which are implemented depend upon the needs of the particular network environment in which the Monitor will operate.

The user invokes the apply button to indicate that the selection process is complete and the type, name, protocol, etc. should be applied. This then updates the screen using the new parameter set that the user selected. The reset button is used to undo the selections and restore them to their values at the last apply operation.

The set of statistics for the selected parameter set is displayed in values panel 604. The members of the sets differ depending upon, for example, what protocol was selected. Figs. 20a-g present examples of the types of statistical variables which are displayed for the DLL, IP, UDP, TCP, ICMP, NFS, and ARP/RARP protocols, respectively. The meaning of the values display fields are described in Appendix I, attached hereto.

Dialogs panel 606 contains a display of the connection statistics for all protocols for a selected node. Within the Management Workstation, connection lists are maintained per node, per supported protocol. When connections are displayed, they are sorted on "Last Seen" with the most current displayed first. A single list returned from the Monitor contains all current connection. For TCP, however, each connection also contains a state and TCP connections are displayed as

Past and Present based upon the returned state of the connection. For certain dialogs, such as TCP and NFS over UDP, there is an associated direction to the dialog, i.e., from the initiator (source) to the receiver (sink). For these dialogs, the direction is identified in a DIR. field. A sample of information that is displayed in dialogs panel 606 is presented in Fig. 21 for current connections.

Values Tool

The values tool provides the user with the ability to look at the statistical database for a network object in detail. When the user invokes this tool, he may select a basic data screen containing a rate values panel 620, a count values panel 622 and a protocols seen panel 626, as shown in Fig. 22, or he may select a traffic matrix screen 628, as illustrated in Fig. 23.

In rate values and count values panels 620 and 622, value tools presents the monitored rate and count statistics, respectively, for a selected protocol. The parameters which are displayed for the different protocols (i.e., different groups) are listed in Appendix II. In general, a data element that is being displayed for a node shows up in three rows, namely, a total for the data element, the number into the data element, and the number out of the data element. Any exceptions to this are identified in Appendix II. Data elements that are displayed for segments, are presented as totals only, with no distinction between Rx and Tx.

When invoked the Values Tool displays a primary screen to the user. The primary screen contains what is considered to be the most significant information for the selected object. The user can view other information for the object (i.e., the statistics for the other parameters) by scrolling down.

The displayed information for the count values and rate values panels 620 and 622 includes the following. An alarm field reports whether an alarm is currently active for this item. It displays as "*" if active alarm is present. A Current Value/Rate field reports the current rate or the value of the counter used to generate threshold alarms for this item. This is reset following each threshold trigger and thus gives an idea of how close to an alarm threshold the variable is. A Typical Value field reports what this item could be expected to read in a "normal" operating situation. This field is filled in for those items where this is predictable and useful. It is maintained in the Workstation database and is modifiable by the user using the set tool. An Accumulated Count field reports the current accumulated value of the item or the current rate. A Max Value field reports the highest value recently seen for the item. This value is reset at intervals defined by a user adjustable parameter (default 30 minutes). This is not a rolling cycle but rather represents the highest value since it was reset which may be from 1 to 30 minutes ago (for a rest period of 30 minutes). It is used only for rates. A Min Value field reports the lowest value recently seen for the item. This operates in the same manner as Max Value field and is used only for rates.

A Percent (%) field reports only for the following variables:

off seg counts:

100(in count / total off seg count)
100(out count / total off seg count)
100(transit count / total off seg count)
100(local count / total off seg count)

off seg rates

100(transit rate / total off seg rate), etc.

protocols

100(frame rate this protocol / total frame rate)

On the right half of the basic display, there the following addtional fields: a High Threshold field and a Sample period for rates field.

Set Tool

The set tool provides the user with the ability to modify the parameters controling the operation of the Monitors and the Management Workstation. These parameters affect both user interface displays and the actual operation of the Monitors. The parameters which can be operated on by the set tool can be divided into the following categories: alarm thresholds, monitoring control, segment Monitor administration, and typical values.

The monitoring control variables specify the actions of the segment Monitors and each Monitor can have a distinct set of control variables (e.g., the parse control records that are described elsewhere). The user is able to define those nodes, segments, dialogs and protocols in which he is interested so as to make the best use of memory space available for data storage. This mechanism allows for load sharing, where mulitple Monitors on the same segment can divide up the total number of network objects which are to be monitored so that no duplication of effort between them takes place.

The monitor administration variables allow the user to modify the operation of the segment Monitor in a more direct manner than the monitoring control variables. Using the set tool, the user can perform those operations such as reset, time changes etc. which are normally the prerogative of a system administrator.

Note that the above descriptions of the tools available through the Management Workstation are not meant to imply that other choices may not be made

regarding the particular information which is displayed and the manner in which it is displayed.

Adaptively Setting Network Monitor Thresholds:

The Workstation sets the thresholds in the Network Monitor based upon the performance of the system as observed over an extended period of time. That is, the Workstation periodically samples the output of the Network Monitors and assembles a model of a normally functioning network. Then, the Workstation sets the thresholds in the Network Monitors based upon that model. If the observation period is chosen to be long enough and since the model represents the "average" of the network performance over the observation period, temporary undesired deviations from normal behavior are smoothed out over time and model tends to accurately reflect normal network behavior.

Referring the Fig. 24, the details of the training procedure for adaptively setting the Network Monitor thresholds are as follows. To begin training, the Workstation sends a start learning command to the Network Monitors from which performance data is desired (step 302). The start learning command disables the thresholds within the Network Monitor and causes the Network Monitor to periodically send data for a predefined set of network parameters to the Management Workstation. (Disabling the thresholds, however, is not necessary. One could have the learning mode operational in parallel with monitoring using existing thresholds.) The set of parameters may be any or all of the previously mentioned parameters for which thresholds are or may be defined.

Throughout the learning period, the Network Monitor sends "snapshots" of the network's performance to the Workstation which, in turn, stores the data in a performance history database 306 (step 304). The network manager sets the length of the learning period.

Typically, it should be long enough to include the full range of load conditions that the network experiences so that a representative performance history is generated. It should also be long enough so that short periods of overload or faulty behavior do not distort the resulting averages.

After the learning period has expired, the network manager, through the Management Workstation, sends a stop learning command to the Monitor (step 308). The Monitor ceases automatically sending further performance data updates to the Workstation and the Workstation processes the data in its performance history database (step 310). The processing may involve simply computing averages for the parameters of interest or it may involve more sophisticated statistical analysis of the data, such as computing means, standard deviations, maximum and minimum values, or using curve fitting to compute rates and other pertinent parameter values.

After the Workstation has statistically analyzed the performance data, it computes a new set of thresholds for the relevant performance parameters (step 312). To do this, it uses formulas which are appropriate to the particular parameter for which a threshold is being computed. That is, if the parameter is one for which one would expect to see wide variations in its value during network monitoring, then the threshold should be set high enough so that the normal expected variations do not trigger alarms. On the other hand, if the parameter is of a type for which only small variations are expected and larger variations indicate a problem, then the threshold should be set to a value that is close to the average observed value. Examples of formulae which may be used to compute thresholds are:

* Highest value seen during learning period;

* Highest value seen during learning period + 10%;
* Highest value seen during learning period + 50%;
* Highest value seen during learning period + user-defined percent;
* Any value of the parameter other than zero;
* Average value seen during learning period + 50%; and
* Average value seen during learning period + user-defined percent.

As should be evident from these examples, there is a broad range of possibilities regarding how to compute a particular threshold. The choice, however, should reflect the parameter's importance in signaling serious network problems and its normal expected behavior (as may be evidenced from the performance history acquired for the parameter during the learning mode).

After the thresholds are computed, the Workstation loads them into the Monitor and instructs the Monitor to revert to normal monitoring using the new thresholds (step 314).

This procedure provides a mechanism enabling the network manager to adaptively reset thresholds in response to changing conditions on the network, shifting usage patterns and evolving network topology. As the network changes over time, the network manager merely invokes the adaptive threshold setting feature and updates the thresholds to reflect those changes.

<u>The Diagnostic Analyzer Module</u>:

The Management Workstation includes a diagnostic analyzer module which automatically detects and diagnoses the existence and cause of certain types of network problems. The functions of the diagnostic module may actually be distributed among the Workstation and the

Network Monitors which are active on the network. In principle, the diagnostic analyzer module includes the following elements for performing its fault detection and analysis functions.

The Management Workstation contains a reference model of a normally operating network. The reference model is generated by observing the performance of the network over an extended period of time and computing averages of the performance statistics that were observed during the observation period. The reference model provides a reference against which future network performance can be compared so as to diagnose and analyze potential problems. The Network Monitor (in particular, the STATS module) includes alarm thresholds on a selected set of the parameters which it monitors. Some of those thresholds are set on parameters which tend to be indicative of the onset or the presence of particular network problems.

During monitoring, when a Monitor threshold is exceeded, thereby indicating a potential problem (e.g. in a TCP connection), the Network Monitor alerts the Workstation by sending an alarm. The Workstation notifies the user and presents the user with the option of either ignoring the alarm or invoking a diagnostic algorithm to analyze the problem. If the user invokes the diagnostic algorithm, the Workstation compares the current performance statistics to its reference model to analyze the problem and report its results. (Of course, this may also be handled automatically so as to not require user intervention.) The Workstation obtains the data on current performance of the network by retrieving the relevant performance statistics from all of the segment Network Monitors that may have information useful to diagnosing the problem.

The details of a specific example involving poor TCP connection performance will now be described. This example refers to a typical network on which the diagnostic analyzer resides, such as the network illustrated in Fig. 25. It includes three segments labelled S1, S2, and S3, a router R1 connecting S1 to S2, a router R2 connecting S2 to S3, and at least two nodes, node A on S1 which communicates with node B on S3. On each segment there is also a Network Monitor 324 to observe the performance of its segment in the manner described earlier. A Management Workstation 320 is also located on S1 and it includes a diagnostic analyzer module 322. For this example, the sympton of the network problem is degraded peformance of a TCP connection between Nodes A and B.

A TCP connection problem may manifest itself in a number of ways, including, for example, excessively high numbers for any of the following:

errors
packets with bad sequence numbers
packets retransmitted
bytes retransmitted
out of order packets
out of order bytes
packets after window closed
bytes after window closed
average and maximum round trip times

or by an unusually low value for the current window size. By setting the appropriate thresholds, the Monitor is programmed to recognize any one or more of these symptons. If any one of of the thresholds is exceeded, the Monitor sends an alarm to the Workstation. The Workstation is programmed to recognize the particular alarm as related to an event which can be further analyzed by its diagnostic analyzer module 322. Thus,

the Workstation presents the user with the option of invoking its diagnostic capabilities (or automatically invokes the diagnostic capabilities).

In general terms, when the diagnostic analyzer is invoked, it looks at the performance data that the segment Monitors produce for the two nodes, for the dialogs between them and for the links that interconnect them and compares that data to the reference model for the network. If a significant divergence from the reference model is identified, the diagnostic analyzer informs the Workstation (and the user) about the nature of the divergence and the likely cause of the problem. In conducting the comparison to "normal" network performance, the network circuit involved in communications between nodes A and B is decomposed into its individual components and diagnostic analysis is performed on each link individually in the effort to isolate the problem further.

The overall structure of the diagnostic algorithm 400 is shown in Fig. 26. When invoked for analyzing a possible TCP problem between nodes A and B, diagnostic analyzer 322 checks for a TCP problem at node A when it is acting as a source node (step 402). To perform this check, diagnostic algorithm 400 invokes a source node analyzer algorithm 450 shown in Fig. 27. If a problem is identified, the Workstation reports that there is a high probability that node A is causing a TCP problem when operating as a source node and it reports the results of the investigation performed by algorithm 450 (step 404).

If node A does not appear to be experiencing a TCP problem when acting as a source node, diagnostic analyzer 322 checks for evidence of a TCP problem at node B when it is acting as a sink node (step 406). To perform this check, diagnostic algorithm 400 invokes a sink node analyzer algorithm 470 shown in Fig. 28. If a problem is

identified, the Workstation reports that there is a high probability that node B is causing a TCP problem when operating as a sink node and it reports the results of the investigation performed by algorithm 470 (step 408).

Note that source and sink nodes are concepts which apply to those dialogs for which a direction of the communication can be defined. For example, the source node may be the one which initiated the dialog for the purpose of sending data to the other node, i.e., the sink node.

If node B does not appear to be experiencing a TCP problem when acting as a sink node, diagnostic analyzer 322 checks for evidence of a TCP problem on the link between Node A and Node B (step 410). To perform this check, diagnostic algorithm 400 invokes a link analysis algorithm 550 shown in Fig. 29. If a problem is identified, the Workstation reports that there is a high probability that a TCP problem exists on the link and it reports the results of the investigation performed by link analysis algorithm 550 (step 412).

If the link does not appear to be experiencing a TCP problem, diagnostic analyzer 322 checks for evidence of a TCP problem at node B when it is acting as a source node (step 414). To perform this check, diagnostic algorithm 400 invokes the previously mentioned source algorithm 450 for Node B. If a problem is identified, the Workstation reports that there is a medium probability that node B is causing a TCP problem when operating as a source node and it reports the results of the investigation performed by algorithm 450 (step 416).

If node B does not appear to be experiencing a TCP problem when acting as a source node, diagnostic analyzer 322 checks for a TCP problem at node A when it is acting as a sink node (step 418). To perform this check, diagnostic algorithm 400 invokes sink node analyzer

algorithm 470 for Node A. If a problem is identified, the Network Monitor reports that there is a medium probability that node A is causing a TCP problem when operating as a sink node and it reports the results of the investigation performed by algorithm 470 (step 420).

Finally, if node A does not appear to be experiencing a TCP problem when acting as a sink node, diagnostic analyzer 322 reports that it was not able to isolate the cause of a TCP problem (step 422).

The algorithms which are called from within the above-described diagnostic algorithm will now be described. Referring to Fig. 27, source node analyzer algorithm 450 checks whether a particular node is causing a TCP problem when operating as a source node. The strategy is as follows. To determine whether a TCP problem exists at this node which is the source node for the TCP connection, look at other connections for which this node is a source. If other TCP connections are okay, then there is probably not a problem with this node. This is an easy check with a high probability of being correct. If no other good connections exist, then look at the lower layers for possible reasons. Start at DLL and work up as problems at lower layers are more fundamental, i.e., they cause problems at higher layers whereas the reverse is not true.

In accordance with this approach, algorithm 450 first determines whether the node is acting as a source node in any other TCP connection and, if so, whether the other connection is okay (step 452). If the node is performing satisfactorily as a source node in another TCP connection, algorithm 450 reports that there is no problem at the source node and returns to diagnostic algorithm 400 (step 454). If algorithm 450 cannot identify any other TCP connections involving this node that are okay, it moves up through the protocol stack

checking each level for a problem. In this case, it then checks for DLL problems at the node when it is acting as a source node by calling an DLL problem checking routine 510 (see Fig. 30) (step 456). If a DLL problem is found, that fact is reported (step 458). If no DLL problems are found, algorithm 450 checks for an IP problem at the node when it is acting as a source by calling an IP problem checking routine 490 (see Fig. 31) (step 460). If an IP problem is found, that fact is reported (step 462). If no IP problems are found, algorithm 450 checks whether any other TCP connection in which the node participates as a source is not okay (step 464). If another TCP connection involving the node exists and it is not okay, algorithm 450 reports a TCP problem at the node (step 466). If no other TCP connections where the node is acting as a source node can be found, algorithm 450 exits.

Referring to Fig. 28, sink node analyzer algorithm 470 checks whether a particular node is causing a TCP problem when operating as a sink node. It first determines whether the node is acting as a sink node in any other TCP connection and, if so, whether the other connection is okay (step 472). If the node is performing satisfactorily as a sink node in another TCP connection, algorithm 470 reports that there is no problem at the source node and returns to diagnostic algorithm 400 (step 474). If algorithm 470 cannot identify any other TCP connections involving this node that are okay, it then checks for DLL problems at the node when it is acting as a sink node by calling DLL problem checking routine 510 (step 476). If a DLL problem is found, that fact is reported (step 478). If no DLL problems are found, algorithm 470 checks for an IP problem at the node when it is acting as a sink by calling IP problem checking routine 490 (step 480). If an IP problem is found, that

fact is reported (step 482). If no IP problems are found, algorithm 470 checks whether any other TCP connection in which the node participates as a sink is not okay (step 484). If another TCP connection involving the node as a sink exists and it is not okay, algorithm 470 reports a TCP problem at the node (step 486). If no other TCP connections where the node is acting as a sink node can be found, algorithm 470 exits.

Referring to Fig. 31, IP problem checking routine 490 checks for IP problems at a node. It does this by comparing the IP performance statistics for the node to the reference model (steps 492 and 494). If it detects any significant deviations from the reference model, it reports that there is an IP problem at the node (step 496). If no significant deviations are noted, it reports that there is no IP problem at the node (step 498).

As revealed by examining Fig. 30, DLL problem checking routine 510 operates in a similar manner to IP problem checking routine 490, with the exception that it examines a different set of parameters (i.e., DLL parameters) for significant deviations.

Referring the Fig. 29, link analysis logic 550 first determines whether any other TCP connection for the link is operating properly (step 552). If a properly operating TCP connection exists on the link, indicating that there is no link problem, link analysis logic 550 reports that the link is okay (step 554). If a properly operating TCP connection cannot be found, the link is decomposed into its constituent components and an IP link component problem checking routine 570 (see Fig. 32) is invoked for each of the link components (step 556). IP link component problem routine 570 evaluates the link component by checking the IP layer statistics for the relevant link component.

The decomposition of the link into its components arranges them in order of their distance from the source node and the analysis of the components proceeds in that order. Thus, for example, the link components which make up the link between nodes A and B include in order: segment S1, router R1, segment S2, router R2, and segment S3. The IP data for these various components are analyzed in the following order:

IP data for segment S1
IP data for address R1
IP data for source node to R1
IP data for S1 to S2
IP data for S2
IP data for address R2
IP data for S3
IP data for S2 to S3
IP data for S1 to S3

As shown in Fig. 32, IP link component problem checking routine 570 compares IP statistics for the link component to the reference model (step 572) to determine whether network performance deviates significantly from that specified by the model (step 574). If significant deviations are detected, routine 570 reports that there is an IP problem at the link component (step 576). Otherwise, it reports that it found no IP problem (step 578).

Referring back to Fig. 29, after completing the IP problem analysis for all of the link components, logic 550 then invokes a DLL link component problem checking routine 580 (see Fig. 33) for each link component to check its DLL statistics (step 558).

DLL link problem routine 580 is similar to IP link problem routine 570. As shown in Fig. 33, DLL link problem checking routine 580 compares DLL statistics for the link to the reference model (step 582) to determine

whether network performance at the DLL deviates significantly from that specified by the model (step 584). If significant deviations are detected, routine 580 reports that there is a DLL problem at the link component (step 586). Otherwise, it reports that no DLL problems were found (step 588).

Referring back to Fig. 29, after completing the DLL problem analysis for all of the link components, logic 550 checks whether there is any other TCP on the link (step 560). If another TCP exists on the link (which implies that the other TCP is also not operating properly), logic 550 reports that there is a TCP problem on the link (step 562). Otherwise, logic 550 reports that there was not enough information from the existing packet traffic to determine whether there was a link problem (step 564)

If the analysis of the link components does not isolate the source of the problem and if there were components for which sufficient information was not available (due possibly to lack of traffic over through that component), the user may send test messages to those components to generate the information needed to evaluate its performance.

The reference model against which comparisons are made to detect and isolate malfunctions may be generated by examining the behavior of the network over an extended period of operation or over multiple periods of operation. During those periods of operation, average values and maximum excursions (or standard deviations) for observed statistics are computed. These values provide an initial estimate of a model of a properly functioning system. As more experience with the network is obtained and as more historical data on the various statistics is accumulated the thresholds for detecting actual malfunctions or imminent malfunctions and the

reference model can be revised to reflect the new experience.

What constitutes a significant deviation from the reference model depends upon the particular parameter involved. Some parameters will not deviate from the expected norm and thus any deviation would be considered to be significant, for example, consider ICMP messages of type "destination unreachable," IP errors, TCP errors. Other parameters will normally vary within a wide range of acceptable values, and only if they move outside of that range should the deviation be considered significant. The acceptable ranges of variation can be determined by watching network performance over a sustained period of operation.

The parameters which tend to provide useful information for identifying and isolating problems at the node level for the different protocols and layers include the following.

<u>TCP</u>
error rate
header byte rate
packets retransmitted
bytes retransmitted
packets after window closed
bytes after window closed

<u>UDP</u>
error rate
header byte rate

<u>IP</u>
error rate
header byte rate
fragmentation rate
all ICMP messages of type destination

unreachable, parameter problem,
redirection

DLL
error rate
runts

For diagnosing network segment problems, the above-identified parameters are also useful with the addition of the alignment rate and the collision rate at the DLL. All or some subset of these parameters may be included among the set of parameters which are examined during the diagnostic procedure to detect and isolate network problems.

The above-described technique can be applied to a wide range of problems on the network, including among others, the following:

TCP Connection fails to establish
UDP Connection performs poorly
UDP not working at all
IP poor performance/high error rate
IP not working at all
DLL poor performance/high error rate
DLL not working at all

For each of these problems, the diagnostic approach would be similar to that described above, using, of course, different parameters to identify the potential problem and isolate its cause.

The Event Timing Module

Referring again to Fig. 5, the RTP is programmed to detect the occurrence of certain transactions for which timing information is desired. The transactions typically occur within a dialog at a particular layer of the protocol stack and they involve a first event (i.e., an initiating event) and a subsequent partner event or response. The events are protocol messages that arrive

at the Network Monitor, are parsed by the RTP and then passed to Event Timing Module (ETM) for processing. A transaction of interest might be, for example, a read of a file on a server. In that case, the initiating event is the read request and the partner event is the read response. The time of interest is the time required to receive a response to the read request (i.e., the transaction time). The transaction time provides a useful measure of network performance and if measured at various times throughout the day under different load conditions gives a measure of how different loads affect network response times. The layer of the communicaton protocol at which the relevant dialog takes place will of course depend upon the nature of the event.

In general, when the RTP detects an event, it transfers control to the ETM which records an arrival time for the event. If the event is an initiating event, the ETM stores the arrival time in an event timing database 300 (see Fig. 34) for future use. If the event is a partner event, the ETM computes a difference between that arrival time and an earlier stored time for the initiating event to determine the complete transaction time.

Event timing database 300 is an array of records 302. Each record 302 includes a dialog field 304 for identifying the dialog over which the transactions of interest are occurring and it includes an entry type field 306 for identifying the event type of interest. Each record 302 also includes a start time field 308 for storing the arrival time of the initiating event and an average delay time field 310 for storing the computed average delay for the transactions. A more detailed description of the operation of the ETM follows.

Referring to Fig. 35, when the RTP detects the arrival of a packet of the type for which timing

information is being kept, it passes control to the ETM along with relevant information from the packet, such as the dialog identifier and the event type (step 320). The ETM then determines whether it is to keep timing information for that particular event by checking the event timing database (step 322). Since each event type can have multiple occurrences (i.e., there can be multiple dialogs at a given layer), the dialog identifier is used to distinguish between events of the same type for different dialogs and to identify those for which information has been requested. All of the dialog/events of interest are identified in the event timing database. If the current dialog and event appear in the event timing database, indicating that the event should be timed, the ETM determines whether the event is a starting event or an ending event so that it may be processed properly (step 324). For certain events, the absence of a start time in the entry field of the appropriate record 302 in event timing database 300 is one indicator that the event represents a start time; otherwise, it is an end time event. For other events, the ETM determines if the start time is to be set by the event type as specified in the packet being parsed. For example, if the event is a file read a start time is stored. If the event is the read completion it represents an end time. In general, each protocol event will have its own intrinsic meaning for how to determine start and end times.

Note that the arrival time is only an estimate of the actual arrival time due to possible queuing and other processing delays. Nevertheless, the delays are generally so small in comparison to the transaction times being measured that they are of little consequence.

In step 324, if the event represents a start time, the ETM gets the current time from the kernal and stores

it in start time field 308 of the appropriate record in event timing database 300 (step 326). If the event represents an end time event, the ETM obtains the current time from the kernel and computes a difference between that time and the corresponding start time found in event timing database 300 (step 328). This represents the total time for the transaction of interest. It is combined with the stored average transaction time to compute a new running average transaction time for that event (step 330).

Any one of many different methods can be used to compute the running average transaction time. For example, the following formula can be used:

New Avg. = [(5 * Stored Avg.) + Transaction Time]/6.

After six transaction have been timed, the computed new average becomes a running average for the transaction times. The ETM stores this computed average in the appropriate record of event timing database 300, replacing the previous average transaction time stored in that record, and it clears start time entry field 308 for that record in preparation for timing the next transaction.

After processing the event in steps 322, 326, and 330, the ETM checks the age of all of the start time entries in the event timing database 300 to determine if any of them are too "old" (step 332). If the difference between the current time and any of the start times exceeds a preselected threshold, indicating that a partner event has not occurred within a reasonable period of time, the ETM deletes the old start time entry for that dialog/event (step 334). This insures that a missed packet for a partner event does not result in an erroneously large transaction time which throws off the running average for that event.

If the average transaction time increases beyond a preselected threshold set for timing events, an alarm is sent to the Workstation.

Two examples will now be described to illustrate the operation of the ETM for specific event types. In the first example, Node A of Fig. 25 is communicating with Node B using the NFS protocol. Node A is the client while Node B is the server. The Network Monitor resides on the same segment as node A, but this is not a requirement. When Node A issues a read request to Node B, the Network Monitor sees the request and the RTP within the Network Monitor transfers control to the ETM. Since it is a read, the ETM stores a start time in the Event Timing Database. Thus, the start time is the time at which the read was initiated.

After some delay, caused by the transmission delays of getting the read message to node B, node B performs the read and sends a response back to node A. After some further transmission delays in returning the read response, the Network Monitor receives the second packet for the event. At the time, the ETM recognizes that the event is an end time event and updates the average transaction time entry in the appropriate record with a new computed running average. The ETM then compares the average transaction time with the threshold for this event and if it has been exceeded, issues an alarm to the Workstation.

In the second example, node A is communicating with Node B using the Telnet protocol. Telnet is a virtual terminal protocol. The events of interest take place long after the initial connection has been established. Node A is typing at a standard ASCII (VT100 class) terminal which is logically (through the network) connected to Node B. Node B has an application which is receiving the characters being typed on Node A and, at

appropriate times, indicated by the logic of the applications, sends characters back to the terminal located on Node A. Thus, every time node A sends characters to B, the Network Monitor sees the transmission.

In this case, there are several transaction times which could provide useful network performance information. They include, for example, the amount of time it takes to echo characters typed at the keyboard through the network and back to the display screen, the delay between typing an end of line command and seeing the completion of the application event come back or the network delays incurred in sending a packet and receiving acknowledgment for when it was received.

In this example, the particular time being measured is the time it takes for the network to send a packet and receive an acknowledgement that the packet has arrived. Since Telnet runs on top of TCP, which in turn runs on top of IP, the Network Monitor monitors the TCP acknowledge end-to-end time delays.

Note that this is a design choice of the implementation and that all events visible to the Network Monitor by virtue of the fact that information is in the packet could be measured.

When Node A transmits a data packet to Node B, the Network Monitor receives the packet. The RTP recognizes the packet as being part of a timed transaction and passes control to the ETM. The ETM recognizes it as a start time event, stores the start time in the event timing database and returns control to the RTP after checking for aging.

When Node B receives the data packet from Node A, it sends back an acknowledgment packet. When the Network Monitor sees that packet, it delivers the event to the ETM, which recognizes it as an end time event. The ETM

calculates the delay time for the complete transaction and uses that to update the average transaction time. The ETM then compares the new average transaction time with the threshold for this event. If it has been exceeded, the ETM issues an alarm to the Workstation.

Note that this example is measuring something very different than the previous example. The first example measures the time it takes to traverse the network, perform an action and return that result to the requesting node. It measures performance as seen by the user and it includes delay times from the network as well as delay times from the File Server.

The second example is measuring network delays without looking at the service delays. That is, the ETM is measuring the amount of time it takes to send a packet to a node and receive the acknowledgement of the receipt of the message. In this example, the ETM is measuring transmissions delays as well as processing delays associated with network traffic, but not anything having to do with non-network processing.

As can be seen from the above examples, the ETM can measure a broad range of events. Each of these events can be measured passively and without the cooperation of the nodes that are actually participating in the transmission.

<u>The Address Tracker Module (ATM)</u>

Address tracker module (ATM) 43, one of the software modules in the Network Monitor (see Fig. 5), operates on networks on which the node addresses for particular node to node connections are assigned dynamically. An Appletalk® Network, developed by Apple Computer Company, is an example of a network which uses dynamic node addressing. In such networks, the dynamic change in the address of a particular service causes difficulty troubleshooting the network because the

network manager may not know where the various nodes are and what they are called. In addition, foreign network addresses (e.g., the IP addresses used by that node for communication over an IP network to which if is connected) can not be relied upon to point to a particular node. ATM 43 solves this problem by passively monitoring the network traffic and collecting a table showing the node address to node name mappings.

In the following description, the network on which the Monitor is located is assumed to be an Appletalk® Network. Thus, as background for the following discussion, the manner in which the dynamic node addressing mechanism operates on that network will first be described.

When a node is activated on the Appletalk® Network, it establishes its own node address in accordance with protocol referred to as the Local Link Access Protocol (LLAP). That is, the node guesses its own node address and then verifies that no other node on the network is using that address. The node verifies the uniqueness of its guess by sending an LLAP Enquiry control packet informing all other nodes on the network that it is going to assign itself a particular address unless another node responds that the address has already been assigned. If no other node claims that address as its own by sending an LLAP acknowledgment control packet, the first node uses the address which it has selected. If another node claims the address as its own, the first node tries another address. This continues until, the node finds an unused address.

When the first node wants to communicate with a second node, it must determine the dynamically assigned node address of the second node. It does this in accordance with another protocol referred to as the Name Binding Protocol (NBP). The Name Binding Protocol is

used to map or bind human understandable node names with machine understandable node addresses. The NBP allows nodes to dynamically translate a string of characters (i.e., a node name) into a node address. The node needing to communicate with another node broadcasts an NBP Lookup packet containing the name for which a node address is being requested. The node having the name being requested responds with its address and returns a Lookup Reply packet containing its address to the original requesting node. The first node then uses that address its current communications with the second node.

Referring to Fig. 36, the network includes an Appletalk® Network segment 702 and a TCP/IP segment 704, each of which are connected to a larger network 706 through their respective gateways 708. A Monitor 710, including a Real Time Parser (RTP) 712 and an Address Tracking Module (ATM) 714, is located on Appletalk network segment 702 along with other nodes 711. A Management Workstation 716 is located on segment 704. It is assumed that Monitor 710 has the features and capabilities previously described; therefore, those features not specifically related to the dynamic node addressing capability will not be repeated here but rather the reader is referred to the earlier discussion. Suffice it to say that Monitor 710 is, of course, adapted to operate on Appletalk Network segment 702, to parse and analyze the packets which are transmitted over that segment according to the Appletalk® family of protocols and to communicate the information which it extracts from the network to Management Workstation 716 located on segment 704.

Within Monitor 710, ATM 714 maintains a name table data structure 730 such as is shown in Fig. 37. Name Table 720 includes records 722, each of which has a node name field 724, a node address field 726, an IP address

field 728, and a time field 729. ATM 714 uses Name Table 720 to keep track of the mappings of node names to node address and to IP address. The relevance of each of the fields of records 722 in Name Table 720 are explained in the following description of how ATM 714 operates.

In general, Monitor 710 operates as previously described. That is, it passively monitors all packet traffic over segment 702 and sends all packets to RTP 712 for parsing. When RTP 712 recognizes an Appletalk packet, it transfers control to ATM 714 which analyzes the packet for the presence of address mapping information.

The operation of ATM 714 is shown in greater detail in the flow diagram of Fig. 38. When ATM 714 receives control from RTP 712, it takes the packet (step 730 and strips off the lower layers of the protocol until it determines whether there is a Name Binding Protocol message inside the packet (step 732). If it is a NBP message, ATM 714 then determines whether it is new name Lookup message (step 734). If it is a new name Lookup message, ATM 714 extracts the name from the message (i.e., the name for which a node address is being requested) and adds the name to the node name field 724 of a record 722 in Name Table 720 (step 736).

If the message is an NBP message but it is not a Lookup message, ATM 714 determines whether it is a Lookup Reply (step 738). If it is a Lookup Reply, signifying that it contains a node name/node address binding, ATM 714 extracts the name and the assigned node address from the message and adds this information to Name Table 720. ATM 714 does this by searching the name fields of records 722 in Name Table 720 until it locates the name. Then, it updates the node address field of the identified record to contain the node address which was extracted from the received NBP packet. ATM 714 also updates time

field 729 to record the time at which the message was processed.

After ATM 714 has updated the address field of the appropriate record, it determines whether any records 722 in Name Table 720 should be aged out (step 742). ATM 714 compares the current time to the times recorded in the time fields. If the elapsed time is greater than a preselected time period (e.g. 48 hours), ATM 714 clears the record of all information (step 744). After that, it awaits the next packet from RTP 712.

As ATM 714 is processing each a packet and it determines either that it does not contain an NBP message (step 732) or it does not contain a Lookup Reply message (step 738), ATM 714 branches to step 742 to perform the age out check before going on to the next packet from RTP 712.

The Appletalk to IP gateways provide services that allow an Appletalk Node to dynamically connect to an IP address for communicating with IP nodes. This service extends the dynamic node address mechanism to the IP world for all Appletalk nodes. While the flexibility provided is helpful to the users, the network manager is faced with the problem of not knowing which Appletalk Nodes are currently using a particular IP address and thus, they can not easily track down problems created by the particular node.

ATM 714 can use passive monitoring of the IP address assignment mechanisms to provide the network manager a Name-to-IP address mapping.

If ATM 714 is also keeping IP address information, it implements the additional steps shown in Fig. 39 after completing the node name to node address mapping steps. ATM 714 again checks whether it is an NBP message (step 748). If it is an NBP message, ATM 714 checks whether it is a response to an IP address request (step 750). IP

address requests are typically implied by an NBP Lookup request for an IP gateway. The gateway responds by supplying the gateway address as well as an IP address that is assigned to the requesting node. If the NBP message is an IP address response, ATM 714 looks up the requesting node in Name Table 720 (step 752) and stores the IP address assignment in the IP address field of the appropriate record 722 (step 754).

After storing the IP address assignment information, ATM 714 locates all other records 722 in Name Table 720 which contain that IP address. Since the IP address has been assigned to a new node name, those old entries are no longer valid and must be eliminated. Therefore, ATM 714 purges the IP address fields of those records (step 756). After doing this cleanup step, ATM 714 returns control to RTP 712.

Other embodiments are within the following claims. For example, the Network Monitor can be adapted to identify node types by analyzing the type of packet traffic to or from the node. If the node being monitored is receiving mount requests, the Monitor would report that the node is behaving like node a file server. If the node is issuing routing requests, the Monitor would report that the node is behaving like a router. In either case, the network manager can check a table of what nodes are permitted to provide what functions to determine whether the node is authorized to function as either a file server or a router, and if not, can take appropriate action to correct the problem.

APPENDIX I

SNMP MIB Subset Supported

This is the subset of the standard MIB which can be obtained by monitoring.

Refer to RFC 1066 Management Information Base for an explanation on the items which follow.

System group:
none

Interfaces group
ifType
ifPhysAddress
ifOperStatus
ifInOctets
ifInUcastPkts
ifInNUcastPkts
ifOutOctets
ifOutUcastPkts
ifOutNUcastPkts

Address Translation group
none

IP group
ipForwarding
ipDefaultTTL
ipInReceives
ipInHdrErrors
ipInAddrErrors
ipForwDatagrams
ipReasmReqds
ipFragCreates

IP Address Table
ipAddress
ipAdEntBcastAddr

IP Routing Table
none

ICMP group
icmpInMsgs
icmpInErrors
icmpInDestUnreachs
icmpInTimeExcds
icmpInParmProbs
icmpInSrcQuenchs
icmpInRedirects
icmpInEchoes

icmpInEchoReps
icmpInTimestamps
icmpInTimestampReps
icmpInAddrMasks
icmpInAddrmaskReps
icmpOutMsgs
imcpOutDestrUnreachs
icmpOutTimeExcds
icmpOutParmProbs
icmpOutSrcQuenchs
icmpOutRedirects
icmpOutEchoes
icmpOutEchoReps
icmpOutTimestamps
icmpOutTimestampReps
icmpOutAddrMasks
icmpOutAddrmaskReps

TCP group
tcpActiveOpens
tcpPassiveOpens
tcpAttempFails
tcpEstabResets
tcpCurrEstab
tcpInSegs
tcpOutSegs
tcpRetransSegs
tcpConnTable

UDP group
udpInDatagrams
udpInErrors
udpOutDatagrams
udpOutErrors

EGP group
egpInMsgs
egpInErrors
egpOutMsgs
egpOutErrors

APPENDIX II

MIB Definitions for Network Monitor

1. Common MIB Definitions

Definitions

```
MIB_BUCKETS_PER_RATE          12
MIB_PROTOCOLS_PER_DIALOG      10
MibBucketsPerRate             12
MibProtocolsPerDialog         10

MIB_MAX_PROTOCOL              10
MIB_MAX_MOST_ACTIVE           5
MIB_MAX_DIALOG                3
```

Structures Used

```
typedef struct {
    Byte                year
    Byte                month
    Byte                date
    Byte                day
    Byte                hour
    Byte                minute
    Byte                second
    Byte                unused
} MibTimeOfDay


typedef struct mib_count32_type {
Uint32          accum           ( Long term accum. count)
Uint32          current         ( Present running count)
Uint32          highThld
} MibCount32


typedef struct mib_count64_type {
Uint64          accum           ( Long term accum. count)
Uint64          current         ( Present running count)
Uint64          highThld
} MibCount64

typedef struct mib_meter_type {
Uint32                          current
Uint32                          high
Uint32                          low
Uint32                          highThld
} MibMeter
typedef struct mib_average_meter_type {
Uint32                          current
```

```
Uint32                      high
Uint32                      low
Uint32                      highThld
} MibAverageMeter


typedef struct mib_percent_type {
     Uint32                      current
     Uint32                      high
     Uint32                      low
     Uint32                      highThld
} MibPercent


typedef struct mib_rolling_rate_type {
     Uint32                      current
     Uint32                      high
     Uint32                      low
     Uint32                      highThld
} MibRollingRate


typedef MibRollingRate MibRatePerS
typedef MibRollingRate MibRatePerH

typedef Uint32 MibShortRatePerS
typedef Uint32 MibShortRatePerM


typedef struct mib_short_count32_type {
Uint32          current         ( Present running count)
Uint32          accum           ( Long term accum. count)
} MibShortCount32


typedef struct mib_bucket_rate_type {
Uint32          current         ( Present rate)
Uint32          rates[MIB_BUCKETS_PER_RATE]( 12 5 minute
count buckets )
Uint32          maxRates[MIB_BUCKETS_PER_RATE]( 12 5-min.
max
rate buckets )
} MibBucketRate
```

Most Active Table Definitions

```
typedef struct mib_most_active_entry_type {
     MibAddress                  address
```

```
    MibCount32                packetCount
    MibRatePerS               packetRate
} MibMostActiveEntry


typedef struct mib_most_active_table_type {
Uint32                    numEntries
Uint32                    nextEntry
MibMostActiveEntry  mostActiveEntry[MIB_MAX_MOST_ACTIVE]
    } MibMostActiveTable
```

Protocol Table Definitions

```
typedef struct mib_protocol_entry_type {
    Uint32                    protocol
    MibCount32                packetCount
    MibRatePerS               packetRate
} MibProtocolEntry


typedef struct mib_protocol_table_type {
    Uint32                    numEntries
    Uint32                    nextEntry
    MibProtocolEntry    protocolEntry[MIB_MAX_PROTOCOL]
    } MibProtocolTable
```

Dialog Table Definitions

```
typedef struct mib_transport_type {
    Uint32              transportProtocol
    Uint32              applicationProtocol
    Uint32              initiator
    Uint32              connectionRetries
    Uint32              addr1_window
    Uint32              addr2_window
    Uint32              state
    Uint32              closeReason
    } MibTransportType


typedef struct mib_dialog_entry_type {
MibAddress                addresses
Uint32                    protocolEntries
Uint32
protocols[MIB_PROTOCOLS_PER_DIALOG]
MibTimeOfDay              gmt
Uint32                    startTime
Uint32                    lastTime
Uint32                    alarmsSent
MibCount32                packets
MibRatePerS               packetRate
```

```
MibCount32          bytes
MibRatePerS         byteRate
MibCount32          errors
MibRatePerS         errorRate
MibCount32          fragments
MibRatePerS         fragmentRate
MibCount32          rexmits
MibRatePerS         rexmitRate
MibCount32          flowCtrls
MibRatePerS         flowCtrlRate
MibTransportType    transport
} MibDialogEntry
```

Values for the initiator field

```
ConnectionInitiatorUnknown  0
ConnectionInitiatorAddr1    1
ConnectionInitiatorAddr2    2
```

Values for the connectionCloseReason field

```
ConnectionCloseUnknown      0
ConnectionCloseFin          1
ConnectionCloseRst          2
```

Values for the connectionState field

```
ConnectionStateUnknown      0
ConnectionStateConnecting   1
ConnectionStateData         2
ConnectionStateClosing      3
ConnectionStateClosed       4
```

```
typedef struct mib_dialog_table_type {
     Uint32                        numEntries
     Uint32                        nextEntry
     MibDialogEntry           dialogEntry[MIB_MAX_DIALOG]

     } MibDialogTable
```

2. Data link layer mib definitions for Network Monitor mib.

2.1 dll Segment -Summary Tool

```
typedef struct {
     MibShortCount32          frames
     MibBucketRate        frameRate
```

```
    MibShortCount32        bytes
    MibBucketRate      byteRate
    MibShortCount32        errors
    MibBucketRate      errorRate
    Uint32                 protocolCount
    Uint32                 mostActiveCount
    Uint32                 pairCount
    MibShortCount32        rcvOffSegs
    MibBucketRate      rcvOffSegRate
    MibShortCount32        xmtOffSegs
    MibBucketRate      xmtOffSegRate
    MibShortCount32        transits
    MibBucketRate      transitRate
    MibShortCount32        bcasts
    MibBucketRate      bcastRate
    MibShortCount32        mcasts
    MibBucketRate      mcastRate
    MibShortCount32        collisions
    MibShortRatePerS   collisionRate
    MibShortCount32        alignmtErrors
    MibShortRatePerS   alignmtErrorRate
} MibDllSegSumStats
```

2.2 dll Segment -Values Tool

```
typedef struct {
    MibCount32          frames
    MibRatePerS         frameRate
    MibCount32          bytes
    MibRatePerS         byteRate
    MibCount32          errors
    MibRatePerS         errorRate
    MibCount32          rcvOffSegs
    MibRatePerS         rcvOffSegRate
    MibCount32          xmtOffSegs
    MibRatePerS         xmtOffSegRate
    MibCount32          transits
    MibRatePerS         transitRate
    MibCount32          bcasts
    MibRatePerS         bcastRate
    MibCount32          mcasts
    MibRatePerS         mcastRate
    MibCount32          collisions
    MibRatePerS         collisionRate
    MibCount32          alignmtErrors
    MibRatePerS         alignmtErrorRate
    MibCount32          enetFrames
    MibRatePerS         enetFrameRate
    MibCount32          llcFrames
    MibRatePerS         llcFrameRate
    MibCount32          runtFrames
    MibRatePerS         runtFrameRate
```

```
} MibDllSegValStats
```

2.3 dll Address - Summary Tool

```
typedef struct {
     MibShortCount32          frames
     MibBucketRate        frameRate
     MibShortCount32          bytes
     MibBucketRate        byteRate
     MibShortCount32          errors
     MibBucketRate        errorRate
     Uint32                   protocolCount
     Uint32                   mostActiveCount
     Uint32                   pairCount
     MibShortCount32          rcvOffSegs
     MibBucketRate        rcvOffSegRate
     MibShortCount32          xmtOffSegs
     MibBucketRate        xmtOffSegRate
     MibShortCount32          xmtBcasts
     MibBucketRate        xmtBcastRate
     MibShortCount32          xmtMcasts
     MibBucketRate        xmtMcastRate
} MibDllAddrSumStats
```

2.4 dll Address- Values Tool

```
typedef struct {
     MibCount32               rcvFrames
     MibRatePerS              rcvFrameRate
     MibCount32               rcvBytes
     MibRatePerS              rcvByteRate
     MibCount32               rcvErrors
     MibRatePerS              rcvErrorRate
     MibCount32               xmtFrames
     MibRatePerS              xmtFrameRate
     MibCount32               xmtBytes
     MibRatePerS              xmtByteRate
     MibCount32               xmtErrors
     MibRatePerS              xmtErrorRate
     MibCount32               xmtBcasts
     MibRatePerS              xmtBcastRate
     MibCount32               xmtMcasts
     MibRatePerS              xmtMcastRate
     MibCount32               rcvOffSegs
     MibRatePerS              rcvOffSegRate
     MibCount32               xmtOffSegs
     MibRatePerS              xmtOffSegRate
     MibCount32               enetFrames
     MibRatePerS              enetFrameRate
     MibCount32               llcFrames
     MibRatePerS              llcFrameRate
```

```
     MibCount32              runtFrames
     MibRatePerS             runtFrameRate
} MibDllAddrValStats
```

3. IP layer mib definitions for Network Monitor mib.

3.1 ip Segment - Summary Tool

```
typedef struct {
     MibShortCount32         pkts
     MibBucketRate           pktRate
     MibShortCount32         bytes
     MibBucketRate           byteRate
     MibShortCount32         errors
     MibBucketRate           errorRate
     Uint32                  protocolCount
     Uint32                  mostActiveCount
     Uint32                  pairCount
     MibShortCount32         rcvOffSegs
     MibBucketRate           rcvOffSegRate
     MibShortCount32         xmtOffSegs
     MibBucketRate           xmtOffSegRate

     MibShortCount32         transits
     MibBucketRate           transitRate
     MibShortCount32         flowCtrls
     MibBucketRate           flowCtrlRate
     MibShortCount32         bcasts
     MibBucketRate           bcastRate
     MibShortCount32         mcasts
     MibBucketRate           mcastRate
     MibShortCount32         frgmts
     MibBucketRate           frgmtRate
} MibIpSegSumStats
```

3.2 ip Segment - Values Tool

```
typedef struct {
     MibCount32              pkts
     MibRatePerS             pktRate
     MibCount32              bytes
     MibRatePerS             byteRate
     MibCount32              errors
     MibRatePerS             errorRate
     MibCount32              rcvOffSegs
     MibRatePerS             rcvOffSegRate
     MibCount32              xmtOffSegs
     MibRatePerS             xmtOffSegRate
     MibCount32              transits
     MibRatePerS             transitRate
```

```
     MibCount32               bcasts
     MibRatePerS              bcastRate
     MibCount32               mcasts
     MibRatePerS              mcastRate
     MibCount32               hdrBytes
     MibRatePerS              hdrByteRate
     MibCount32               frgmts
     MibRatePerS              frgmtRate
} MibIpSegValStats
```

3.3 ip Address - Summary Tool

```
typedef struct {
     MibShortCount32          pkts
     MibBucketRate            pktRate
     MibShortCount32          bytes
     MibBucketRate            byteRate
     MibShortCount32          errors
     MibBucketRate            errorRate
     Uint32                   protocolCount
     Uint32                   mostActiveCount
     Uint32                   pairCount
     MibShortCount32          rcvOffSegs
     MibBucketRate            rcvOffSegRate
     MibShortCount32          xmtOffSegs
     MibBucketRate            xmtOffSegRate
     MibShortCount32          flowCtrls
     MibBucketRate            flowCtrlRate
     MibShortCount32          frgmts
     MibBucketRate            frgmtRate
     MibShortCount32          xmtBcasts
     MibBucketRate            xmtBcastRate
     MibShortCount32          xmtMcasts
     MibBucketRate            xmtMcastRate
} MibIpAddrSumStats
```

3.4 ip Address - Values Tool

```
typedef struct {
     MibCount32               rcvPkts
     MibRatePerS              rcvPktRate
     MibCount32               rcvBytes
     MibRatePerS              rcvByteRate
     MibCount32               rcvErrors
     MibRatePerS              rcvErrorRate
     MibCount32               xmtPkts
     MibRatePerS              xmtPktRate
     MibCount32               xmtBytes
     MibRatePerS              xmtByteRate
     MibCount32               xmtErrors
     MibRatePerS              xmtErrorRate
     MibCount32               rcvHdrBytes
     MibRatePerS              rcvHdrByteRate
```

```
    MibCount32              xmtHdrBytes
    MibRatePerS             xmtHdrByteRate
    MibCount32              rcvFrgmts
    MibRatePerS             rcvFrgmtRate
    MibCount32              xmtFrgmts
    MibRatePerS             xmtFrgmtRate
    MibCount32              xmtBcasts
    MibRatePerS             xmtBcastRate
    MibCount32              xmtMcasts
    MibRatePerS             xmtMcastRate
    MibCount32              rcvOffSegs
    MibRatePerS             rcvOffSegRate
    MibCount32              xmtOffSegs
    MibRatePerS             xmtOffSegRate
} MibIpAddrValStats
```

4. ICMP layer mib definitions for Network Monitor mib.

4.1 icmp Segment - Summary Tool

```
typedef struct {
    MibShortCount32             pkts
    MibBucketRate           pktRate

    MibShortCount32             bytes
    MibBucketRate           byteRate

    MibShortCount32             errors
    MibBucketRate           errorRate

    Uint32                      mostActiveCount
    Uint32                      pairCount

    MibShortCount32             rcvOffSegs
    MibBucketRate           rcvOffSegRate
    MibShortCount32             xmtOffSegs
    MibBucketRate           xmtOffSegRate
    MibShortCount32             transits
    MibBucketRate           transitRate

    MibShortCount32             echoReq
    MibShortCount32             echoReply
    MibShortCount32             destUnr
    MibShortCount32             srcQuench
    MibShortCount32             redir
    MibShortCount32             timeExceeded
    MibShortCount32             paramProblem
    MibShortCount32             timestampReq
    MibShortCount32             timestampReply
    MibShortCount32             addrMaskReq
    MibShortCount32             addrMaskReply
} MibIcmpSegSumStats
```

4.2 icmp Segment - Values Tool

```
typedef struct {
     MibCount32                 pkts
     MibRatePerS                pktRate

     MibCount32                 bytes
     MibRatePerS                byteRate

     MibCount32                 errors
     MibRatePerS                errorRate

     MibCount32                 rcvOffSegs
     MibRatePerS                rcvOffSegRate
     MibCount32                 xmtOffSegs
     MibRatePerS                xmtOffSegRate
     MibCount32                 transits
     MibRatePerS                transitRate

     MibCount32                 echoReq
     MibRatePerS                echoReqRate
     MibCount32                 echoReply
     MibRatePerS                echoReplyRate

     MibCount32                 destUnrNet
     MibRatePerS                destUnrNetRate
     MibCount32                 destUnrHost
     MibRatePerS                destUnrHostRate
     MibCount32                 destUnrProtocol
     MibRatePerS                destUnrProtocolRate
     MibCount32                 destUnrPort
     MibRatePerS                destUnrPortRate
     MibCount32                 destUnrFrgmt
     MibRatePerS                destUnrFrgmtRate
     MibCount32                 destUnrSrcRoute
     MibRatePerS                destUnrSrcRouteRate
     MibCount32                 destUnrNetUnknown
     MibRatePerS                destUnrNetUnknownRate
     MibCount32                 destUnrHostUnknown
     MibRatePerS                destUnrHostUnknownRate
     MibCount32                 destUnrSrcHostIsolated

     MibRatePerS                destUnrSrcHostIsolatedRate
     MibCount32                 destUnrNetProhibited

     MibRatePerS                destUnrNetProhibitedRate
     MibCount32                 destUnrHostProhibited
     MibRatePerS                destUnrHostProhibitedRate
     MibCount32                 destUnrNetTos
     MibRatePerS                destUnrNetTosRate
     MibCount32                 destUnrHostTos
```

```
    MibRatePerS              destUnrHostTosRate

    MibCount32               srcQuench
    MibRatePerS              srcQuenchRate

    MibCount32               redirNet
    MibRatePerS              redirNetRate
    MibCount32               redirHost
    MibRatePerS              redirHostRate
    MibCount32               redirNetTos
    MibRatePerS              redirNetTosRate
    MibCount32               redirHostTos
    MibRatePerS              redirHostTosRate

    MibCount32               timeExceededInTransit
    MibRatePerS              timeExceededInTransitRate
    MibCount32               timeExceededInReass
    MibRatePerS              timeExceededInReassRate

    MibCount32               paramProblem
    MibRatePerS              paramProblemRate
    MibCount32               paramProblemOption
    MibRatePerS              paramProblemOptionRate

    MibCount32               timestampReq
    MibRatePerS              timestampReqRate
    MibCount32               timestampReply
    MibRatePerS              timestampReplyRate

    MibCount32               addrMaskReq
    MibRatePerS              addrMaskReqRate
    MibCount32               addrMaskReply
    MibRatePerS              addrMaskReplyRate

} MibIcmpSegValStats
```

4.3 icmp Address - Summary Tool

```
typedef struct {
    MibShortCount32          pkts
    MibBucketRate        pktRate

    MibShortCount32          bytes
    MibBucketRate        byteRate

    MibShortCount32          errors
    MibBucketRate        errorRate
    Uint32                   mostActiveCount
    Uint32                   pairCount

    MibShortCount32          rcvOffSegs
    MibBucketRate        rcvOffSegRate
```

```
    MibShortCount32         xmtOffSegs
    MibBucketRate         xmtOffSegRate

    MibShortCount32         echoReq
    MibShortCount32         echoReply
    MibShortCount32         destUnr
    MibShortCount32         srcQuench
    MibShortCount32         redir
    MibShortCount32         paramProblem
    MibShortCount32         timeExceeded
    MibShortCount32         timestampReq
    MibShortCount32         timestampReply
    MibShortCount32         addrMaskReq
    MibShortCount32         addrMaskReply
} MibIcmpAddrSumStats
```

4.4 icmp Address- Values Tool

```
typedef struct {

    MibCount32          rcvPkts
    MibRatePerS         rcvPktRate
    MibCount32          rcvBytes
    MibRatePerS         rcvByteRate
    MibCount32          rcvErrors
    MibRatePerS         rcvErrorRate

    MibCount32          xmtPkts
    MibRatePerS         xmtPktRate
    MibCount32          xmtBytes
    MibRatePerS         xmtByteRate
    MibCount32          xmtErrors
    MibRatePerS         xmtErrorRate

    MibCount32          rcvOffSegs
    MibRatePerS         rcvOffSegRate
    MibCount32          xmtOffSegs
    MibRatePerS         xmtOffSegRate

    MibCount32          rcvDestUnrNet
    MibRatePerS         rcvDestUnrNetRate
    MibCount32          rcvDestUnrHost
    MibRatePerS         rcvDestUnrHostRate
    MibCount32          rcvDestUnrProtocol
    MibRatePerS         rcvDestUnrProtocolRate
    MibCount32          rcvDestUnrPort
    MibRatePerS         rcvDestUnrPortRate
    MibCount32          rcvDestUnrFrgmt
    MibRatePerS         rcvDestUnrFrgmtRate
    MibCount32          rcvDestUnrSrcRoute
    MibRatePerS         rcvDestUnrSrcRouteRate
    MibCount32          rcvDestUnrNetUnknown
```

MibRatePerS	rcvDestUnrNetUnknownRate
MibCount32	rcvDestUnrHostUnknown
MibRatePerS	rcvDestUnrHostUnknownRate
MibCount32	rcvDestUnrSrcHostIsolated
MibRatePerS	rcvDestUnrSrcHostIsolatedRate
MibCount32	rcvDestUnrNetProhibited
MibRatePerS	rcvDestUnrNetProhibitedRate
MibCount32	rcvDestUnrHostProhibited
MibRatePerS	rcvDestUnrHostProhibitedRate
MibCount32	rcvDestUnrNetTos
MibRatePerS	rcvDestUnrNetTosRate
MibCount32	rcvDestUnrHostTos
MibRatePerS	rcvDestUnrHostTosRate
MibCount32	rcvTimeExceededInTransit
MibRatePerS	rcvTimeExceededInTransitRate
MibCount32	rcvTimeExceededInReass
MibRatePerS	rcvTimeExceededInReassRate
MibCount32	rcvParamProblem
MibRatePerS	rcvParamProblemRate
MibCount32	rcvParamProblemOption
MibRatePerS	rcvParamProblemOptionRate
MibCount32	rcvSrcQuench
MibRatePerS	rcvSrcQuenchRate
MibCount32	rcvRedirNet
MibRatePerS	rcvRedirNetRate
MibCount32	rcvRedirHost
MibRatePerS	rcvRedirHostRate
MibCount32	rcvRedirNetTos
MibRatePerS	rcvRedirNetTosRate
MibCount32	rcvRedirHostTos
MibRatePerS	rcvRedirHostTosRate
MibCount32	rcvEchoReq
MibRatePerS	rcvEchoReqRate
MibCount32	rcvEchoReply
MibRatePerS	rcvEchoReplyRate
MibCount32	rcvTimestampReq
MibRatePerS	rcvTimestampReqRate
MibCount32	rcvTimestampReply
MibRatePerS	rcvTimestampReplyRate
MibCount32	rcvAddrMaskReq
MibRatePerS	rcvAddrMaskReqRate
MibCount32	rcvAddrMaskReply
MibRatePerS	rcvAddrMaskReplyRate

MibCount32	xmtDestUnrNet
MibRatePerS	xmtDestUnrNetRate
MibCount32	xmtDestUnrHost
MibRatePerS	xmtDestUnrHostRate
MibCount32	xmtDestUnrProtocol
MibRatePerS	xmtDestUnrProtocolRate
MibCount32	xmtDestUnrPort
MibRatePerS	xmtDestUnrPortRate
MibCount32	xmtDestUnrFrgmt
MibRatePerS	xmtDestUnrFrgmtRate
MibCount32	xmtDestUnrSrcRoute
MibRatePerS	xmtDestUnrSrcRouteRate
MibCount32	xmtDestUnrNetUnknown
MibRatePerS	xmtDestUnrNetUnknownRate
MibCount32	xmtDestUnrHostUnknown
MibRatePerS	xmtDestUnrHostUnknownRate
MibCount32	xmtDestUnrSrcHostIsolated
MibRatePerS	xmtDestUnrSrcHostIsolatedRate
MibCount32	xmtDestUnrNetProhibited
MibRatePerS	xmtDestUnrNetProhibitedRate
MibCount32	xmtDestUnrHostProhibited
MibRatePerS	xmtDestUnrHostProhibitedRate
MibCount32	xmtDestUnrNetTos
MibRatePerS	xmtDestUnrNetTosRate
MibCount32	xmtDestUnrHostTos
MibRatePerS	xmtDestUnrHostTosRate
MibCount32	xmtTimeExceededInTransit
MibRatePerS	xmtTimeExceededInTransitRate
MibCount32	xmtTimeExceededInReass
MibRatePerS	xmtTimeExceededInReassRate
MibCount32	xmtParamProblem
MibRatePerS	xmtParamProblemRate
MibCount32	xmtParamProblemOption
MibRatePerS	xmtParamProblemOptionRate
MibCount32	xmtSrcQuench
MibRatePerS	xmtSrcQuenchRate
MibCount32	xmtRedirNet
MibRatePerS	xmtRedirNetRate
MibCount32	xmtRedirHost
MibRatePerS	xmtRedirHostRate
MibCount32	xmtRedirNetTos
MibRatePerS	xmtRedirNetTosRate
MibCount32	xmtRedirHostTos
MibRatePerS	xmtRedirHostTosRate
MibCount32	xmtEchoReq
MibRatePerS	xmtEchoReqRate
MibCount32	xmtEchoReply

```
    MibRatePerS         xmtEchoReplyRate

    MibCount32          xmtTimestampReq
    MibRatePerS         xmtTimestampReqRate
    MibCount32          xmtTimestampReply
    MibRatePerS         xmtTimestampReplyRate

    MibCount32          xmtAddrMaskReq
    MibRatePerS         xmtAddrMaskReqRate
    MibCount32          xmtAddrMaskReply
    MibRatePerS         xmtAddrMaskReplyRate
}
```

5. TCP layer mib definitions for Network Monitor mib.

5.1 tcp Segment - Summary Tool

```
typedef struct {

    MibShortCount32         pkts
    MibBucketRate       pktRate
    MibShortCount32         bytes
    MibBucketRate       byteRate

    MibShortCount32         errors
    MibBucketRate       errorRate

    Uint32                  protocolCount
    Uint32                  mostActiveCount
    Uint32                  pairCount

    MibShortCount32         rcvOffSegs
    MibBucketRate       rcvOffSegRate
    MibShortCount32         xmtOffSegs
    MibBucketRate       xmtOffSegRate
    MibShortCount32         transits
    MibBucketRate       transitRate

    MibShortCount32         flowCtrls
    MibBucketRate       flowCtrlRate

    MibShortCount32         frgmts
    MibBucketRate       frgmtRate

    MibShortCount32         rexmts
    MibBucketRate       rexmtRate

} MibTcpSegSumStats
```

5.2 tcp Segment - Values Tool

```
typedef struct {

      MibCount32            pkts
      MibRatePerS           pktRate

      MibCount32            bytes
      MibRatePerS           byteRate

      MibCount32            errors
      MibRatePerS           errorRate

      MibCount32            rcvOffSegs
      MibRatePerS           rcvOffSegRate
      MibCount32            xmtOffSegs
      MibRatePerS           xmtOffSegRate
      MibCount32            transits
      MibRatePerS           transitRate

      MibCount32            hdrBytes
      MibRatePerS           hdrByteRate
      MibCount32            frgmts
      MibRatePerS           frgmtRate

      MibCount32            flowCtrls
      MibRatePerS           flowCtrlRate

      MibCount32            rexmts
      MibRatePerS           rexmtRate

      MibCount32            rexmtBytes
      MibRatePerS           rexmtByteRate

      MibCount32            keepAlives
      MibRatePerS           keepAliveRate

      MibCount32            windowProbes
      MibRatePerS           windowProbeRate

      MibCount32            outOfOrder
      MibRatePerS           outOfOrderRate

      MibCount32            afterWindow
      MibRatePerS           afterWindowRate

      MibCount32            afterClose
      MibRatePerS           afterCloseRate

      MibCount32            urgs
      MibRatePerS           urgRate

      MibCount32            rsts
      MibRatePerS           rstRate
```

```
    MibCount32          successfulConnections
     MibRatePerH         successfulConnectionRate
     MibCount32          connectionRetries
     MibRatePerH         connectionRetryRate
     MibCount32          failedConnections
     MibRatePerH         failedConnectionRate
     MibCount32          activeConnections
} MibTcpSegValStats
```

5.3 tcp Address - Summary Tool

```
typedef struct {

     MibShortCount32          pkts
     MibBucketRate       pktRate

     MibShortCount32          bytes
     MibBucketRate       byteRate

     MibShortCount32          errors
     MibBucketRate       errorRate

     Uint32                   protocolCount
     Uint32                   mostActiveCount
     Uint32                   pairCount

     MibShortCount32          rcvOffSegs
     MibBucketRate       rcvOffSegRate
     MibShortCount32          xmtOffSegs
     MibBucketRate       xmtOffSegRate

     MibShortCount32          flowCtrls
     MibBucketRate       flowCtrlRate

     MibShortCount32          frgmts
     MibBucketRate       frgmtRate

     MibShortCount32          rexmts
     MibBucketRate       rexmtRate

} MibTcpAddrSumStats
```

5.4 tcp Address- Values Tool

```
typedef struct {

     MibCount32               rcvPkts
     MibRatePerS              rcvPktRate
     MibCount32               xmtPkts
     MibRatePerS              xmtPktRate
```

```
MibCount32          rcvBytes
MibRatePerS         rcvByteRate
MibCount32          xmtBytes
MibRatePerS         xmtByteRate

MibCount32          rcvErrors
MibRatePerS         rcvErrorRate
MibCount32          xmtErrors
MibRatePerS         xmtErrorRate

MibCount32          rcvOffSegs
MibRatePerS         rcvOffSegRate
MibCount32          xmtOffSegs
MibRatePerS         xmtOffSegRate

MibCount32          rcvHdrBytes
MibRatePerS         rcvHdrByteRate
MibCount32          xmtHdrBytes
MibRatePerS         xmtHdrByteRate

MibCount32          rcvFrgmts
MibRatePerS         rcvFrgmtRate
MibCount32          xmtFrgmts
MibRatePerS         xmtFrgmtRate

MibCount32          rcvRexmts
MibRatePerS         rcvRexmtRate
MibCount32          xmtRexmts
MibRatePerS         xmtRexmtRate

MibCount32          rcvRexmtBytes
MibRatePerS         rcvRexmtByteRate
MibCount32          xmtRexmtBytes
MibRatePerS         xmtRexmtByteRate

MibCount32          rcvKeepAlives
MibRatePerS         rcvKeepAliveRate
MibCount32          xmtKeepAlives
MibRatePerS         xmtKeepAliveRate

MibCount32          rcvWindowProbes
MibRatePerS         rcvWindowProbeRate
MibCount32          xmtWindowProbes
MibRatePerS         xmtWindowProbeRate

MibCount32          rcvOutOfOrder
MibRatePerS         rcvOutOfOrderRate
MibCount32          xmtOutOfOrder
MibRatePerS         xmtOutOfOrderRate

MibCount32          rcvAfterWindow
MibRatePerS         rcvAfterWindowRate
```

```
MibCount32              xmtAfterWindow
MibRatePerS             xmtAfterWindowRate

MibCount32              rcvAfterClose
MibRatePerS             rcvAfterCloseRate
MibCount32              xmtAfterClose
MibRatePerS             xmtAfterCloseRate

MibCount32              rcvUrgs
MibRatePerS             rcvUrgRate
MibCount32              xmtUrgs
MibRatePerS             xmtUrgRate

MibCount32              rcvRsts
MibRatePerS             rcvRstRate
MibCount32              xmtRsts
MibRatePerS             xmtRstRate

MibCount32              successfulConnections
MibRatePerH             successfulConnectionRate
MibCount32              connectionRetries
MibRatePerH             connectionRetryRate
MibCount32              failedConnections
MibRatePerH             failedConnectionRate
MibCount32              activeConnections
```

6. UDP layer mib definitions for Network Monitor mib.

6.1 udp Segment -Summary Tool

```
typedef struct {
    MibShortCount32         pkts
    MibBucketRate       pktRate
    MibShortCount32         bytes
    MibBucketRate       byteRate
    MibShortCount32         errors
    MibBucketRate       errorRate
    MibShortCount32         protocolCount
    MibShortCount32         mostActiveCount
    MibShortCount32         pairCount
    MibShortCount32         rcvOffSegs
    MibBucketRate       rcvOffSegRate
    MibShortCount32         xmtOffSegs
    MibBucketRate       xmtOffSegRate
    MibShortCount32         transits
    MibBucketRate       transitRate
    MibShortCount32         flowCtrls
    MibBucketRate       flowCtrlRate
} MibUdpSegSumStats
```

6.2 udp Segment - Values Tool

```
typedef struct {
     MibCount32               pkts
     MibRatePerS              pktRate
     MibCount32               bytes
     MibRatePerS              byteRate
     MibCount32               errors
     MibRatePerS              errorRate
     MibShortCount32          protocolCount
     MibShortCount32          mostActiveCount
     MibShortCount32          pairCount
     MibCount32               rcvOffSegs
     MibRatePerS              rcvOffSegRate
     MibCount32               xmtOffSegs
     MibRatePerS              xmtOffSegRate
     MibCount32               transits
     MibRatePerS              transitRate
     MibCount32               flowCtrls
     MibRatePerS              flowCtrlRate
     MibCount32               hdrBytes
     MibRatePerS              hdrByteRate
} MibUdpSegValStats
```

6.3 udp Address - Summary Tool

```
typedef struct {
     MibShortCount32          pkts
     MibBucketRate       pktRate
     MibShortCount32          bytes
     MibBucketRate       byteRate
     MibShortCount32          errors
     MibBucketRate       errorRate
     MibShortCount32          protocolCount
     MibShortCount32          mostActiveCount
     MibShortCount32          pairCount
     MibShortCount32          rcvOffSegs
     MibBucketRate       rcvOffSegRate
     MibShortCount32          xmtOffSegs
     MibBucketRate       xmtOffSegRate
     MibShortCount32          flowCtrls
     MibBucketRate       flowCtrlRate
} MibUdpAddrSumStats
```

6.4 udp Address- Values Tool

```
typedef struct {
     MibCount32          rcvPkts
     MibRatePerS         rcvPktRate
     MibCount32          rcvBytes
```

```
     MibRatePerS          rcvByteRate
     MibCount32           rcvErrors
     MibRatePerS          rcvErrorRate
     MibCount32           xmtPkts
     MibRatePerS          xmtPktRate
     MibCount32           xmtBytes
     MibRatePerS          xmtByteRate
     MibCount32           xmtErrors
     MibRatePerS          xmtErrorRate
     MibCount32           rcvHdrBytes
     MibRatePerS          rcvHdrByteRate
     MibCount32           xmtHdrBytes
```

7. Monitor mib definitions for Network Monitor mib.

```
typedef struct {
     int                       length
     char                 no[80]
} MibPhoneNumber

typedef struct {
     MacAddress                lanMacAddr
     IpAddress            lanIpAddr
     Uint32                    lanTftpTimeout
     Uint32                    lanTftpRetryLimit
     Uint32                    lanSnmpTimeout
     Uint32                    lanSnmpRetryLimit
     MibPhoneNumber       serialPhoneNo
     IpAddress            serialIpAddr
     Uint32                    serialTftpTimeout
     Uint32                    serialTftpRetryLimit
     Uint32                    serialSnmpTimeout
     Uint32                    serialSnmpRetryLimit
} MibWsParameters

typedef struct {
     MibAddress                address
     Uint32                    flags
     MibDeviceType        type
     Uint32                    parseControl
} MibParseControl

typedef struct {
     Uint32          numEntries
     Uint32          nextEntry
     MibParseControl mibParseControl[MIB_MAX_PCR]
     } MibParseControlOpaque


typedef struct {
     MacAddress                macAddr
     Byte                 data[256]
```

```
    Uint32              length
derived
} MibAutoTopology
```

7.1 Monitor Control Group

```
typedef struct {
    Uint32              monReset
    MibTimeOfDay        monTOD
    Uint32              trapPermit
    Uint32              dupAddrTrapPermit
    Uint32              newNodeTrapPermit
    Uint32              shakeTime
    Uint32              wsMonLink
    Uint32              minTrapInterval
    Uint32              runMonitor
    MibWsParameters     primaryWsParams
    MibWsParameters     secondaryWsParams
    Uint32              debugLevel
    Uint32              parseCtrl
    Uint32              monitorSegment
    MibAutoTopology     autoTopology
    } MibMonitorControl
```

7.2 Monitor Statistics Group

```
typedef struct {
    MibCount32          dllDropped
    MibRatePerS         dllDroppedRate
    MibCount32          ipDropped
    MibRatePerS         ipDroppedRate
    MibCount32          icmpDropped
    MibRatePerS         icmpDroppedRate
    MibCount32          tcpDropped
    MibRatePerS         tcpDroppedRate
    MibCount32          udpDropped
    MibRatePerS         udpDroppedRate
    MibCount32          arpDropped
    MibRatePerS         arpDroppedRate
    MibCount32          nfsDropped
    MibRatePerS         nfsDroppedRate
    MibCount32          dbProblem
    MibShortCount32     cpuUtilization
    MibShortCount32     memoryUtilization
```

8. Alarm Mib Definitions

8.1 Counter alarm structure

```
typedef struct {
     Uint32                alarm_class
     MibTimeOfDay          gmt
     Uint32                     time_ticks
     MibAddress                 mon_address
     MibAddress                 address
     Uint32                     type
     Uint32                     number
     MibCount32                 value
     Uint32                     user_data_length

      OPTIONAL
     Byte                  user_data[MAX_ALARM_DATA]

OPTIONAL
} MibAlarmCounter
```

8.2 Rate alarm structure

```
typedef struct {
     Uint32                alarm_class
     MibTimeOfDay          gmt
     Uint32                     time_ticks
     MibAddress                 mon_address
     MibAddress                 address
     Uint32                     type
     Uint32                     number
     MibRollingRate        value
     Uint32                     rate_type
     Uint32                     user_data_length

      OPTIONAL
     Byte                  user_data[MAX_ALARM_DATA]

OPTIONAL
} MibAlarmRate
```

8.3 Power-up alarm structure

```
typedef struct {
     Uint32                alarm_class
     MibTimeOfDay          gmt
     Uint32                     time_ticks
     MibAddress                 mon_address
     Uint32                     alarm_reason
     Uint32                     load_type
     Uint32                     cpu_hw_rev
     Uint32                     mon_link_hw_rev
```

```
    Uint32                  mgmt_link_hw_rev
    MibPhoneNumber      mon_phone_no
    Uint32                  error_type
    Uint32                  error_code
    Uint32                  error_param_1
    Uint32                  error_param_2
    Uint32                  error_param_3
} MibAlarmPowerUp
```

8.4 Link-up alarm structure

```
typedef struct {
    Uint32              alarm_class
    MibTimeOfDay        gmt
    Uint32                  time_ticks
    MibAddress              mon_address
    Uint32                  alarm_reason
    Uint32                  load_type
    Uint32                  cpu_hw_rev
    Uint32                  mon_link_hw_rev
    Uint32                  mgmt_link_hw_rev
    MibPhoneNumber      mon_phone_no
    Uint32                  error_type
    Uint32                  error_code
    Uint32                  error_param_1
    Uint32                  error_param_2
    Uint32                  error_param_3
} MibAlarmLinkUp
```

8.5 New node alarm structure

```
typedef struct {
    Uint32              alarm_class
    MibTimeOfDay        gmt
    Uint32                  time_ticks
    MibAddress              mon_address
    MibAddress              node_address
} MibAlarmNewNode
```

APPENDIX III

PROTOCOL VARIABLES

The following is a list of some of the network variables for which data is gathered by the Monitor and a brief explanation of the variable, where appropriate.

DLL Variables

Frames

A frame is a series of bytes with predefined bit sequences that mark the frame's beginning and ending points. A DLL (data link layer) entity sends a message by putting it in a frame and transmitting it on the physical network. It's called a frame because the beginning and ending bit sequences "frame" the message.

Enclosed within the frame are the messages built by higher level protocols, such as IP and UDP. For example, an IP datagram must be placed in a frame before it can be transmitted.

Ethernet frames range from 64 to 1518 bytes in length.

Bytes

Monitor maintains a count and rate for bytes transmitted and received by all monitored objects. For example, for any node, you can monitor the number of bytes in or out to measure the traffic load with respect to that node. For a segment, you can monitor the number of bytes in and out of all nodes on the segment.

Error Frames

A DLL Error Frame is logged in the following cases:

* If the frame is Ethernet, none are logged.
* If the frame is IEEE 802.3:
    - Value of length parameter in header less than 3.

Alignment Errors

The number of frames observed for the selected segment with alignment errors. An alignment error is a frame with a length that is not an exact multiple of 8 bits. The following variables are available only for segments.

Collisions

The number of collisions observed on the selected segment. A collision occurs when two stations attempt to transmit simultaneously. A certain number of collisions are normal. The following variables are available only for segments.

A higher than typical value can mean that the physical interface for a single station has malfunctioned and in not following the protocol.

Broadcast frame

A broadcast frame is a special frame that is received by all stations on the network. Common uses for broadcast frames include ARP (Address Resolution Protocol) and network testing.

Multicast Frame

A multicast frame is a special frame that is received by a predetermined set of stations. Multicasting is used to send a message to a set of stations using a single frame, thus reducing network loading.

Off-segment

Off-segment frames are frames that the Monitor observes on the local segment, but are destined for or originated by nodes not on the local segment. All off-segment frames then are either routed to, from, or across the local segment.

Off-segment variables

Off-segment variables are a measure of the amount of routing or bridging that is occurring. Excessive off-segment traffic may mean that certain nodes on one segment are communicating primarily with nodes on other segments. If you identify these nodes and move them to the segments where their primary communications partners are, you may lessen the overall loading on your network.

Off-segment Transit Frames

The number of frames observed on the selected segment not into or out of a node on the selected segment. For these frames, the selected segment is an intermediate hop in a route between the originating and destination

segments. (This variable applies only to segments, not to nodes.)

IP Variables

IP Packets

An IP packet or datagram is a string of bytes that is transferred as a unit across the IP network. It has two parts: the IP header, which contains control information such as the source and destination IP addresses; and the data to be transferred to the destination user.

Bytes

The Monitor maintains a count and rate for bytes into and out of all monitored objects. For example, you can monitor the number of bytes into or out of a chosen node to measure the traffic load with respect to that node. You can monitor the number of bytes into and out of all nodes on the segment.

IP Error Packets

An IP error packet is logged when the monitor observes a packet with an error in its IP header. Possible errors are as follows:

* IP header length is less than 20 bytes
* IP header length is greater than the length of the IP packet
* Packet length is less than the IP header length.
* If offset is set for fragmentation, but the frame should not be fragmented.

IP Fragments

If an IP datagram is too large to pass through a subnetwork or router, the IP router that is transmitting the original datagram divides it into fragment datagrams. The destination station reassembles the original datagram once it has received all the fragments.

Fragmentation usually occurs because packets are being routed through a network segment that has physical technology or configuration that restricts the IP datagram size to one smaller that the IP datagram size used on the originating segment.

For example, the maximum frame size in an IEEE 802.5 physical network is 16000 octets, whereas the maximum frame size on an Ethernet physical network is about 1500 octets. In this case, a large frame originating on the IEEE 802.5 network would have to be divided into many fragments before it could be transmitted onto the Ethernet network.

Note that a fragment is a complete and correct IP datagram. Do not confuse IP fragments with the Ethernet fragment errors.

Higher than typical values for these parameters may mean that one or more commonly-used communications routes are forcing fragmentation to occur.

Example: new nodes have been added that access a server across a fragmenting route. The number of additional packets causes delays on the server's segment. The solution is to reconnect the new nodes to a different segment that has a non-fragmenting route to the server.

IP Header Bytes

The header is the portion of the IP packet that contains control information used by the protocol, such as source and destination IP addresses.

Broadcast and Multicast packets

A broadcast packet is special packet that is received by all stations on the network.

A multicast packet is a packet that is received by a predefined set of stations. Multicasting is used to send a message to a set of stations using a single packet.

IP Off-segment Packets

Off-segment packets are packets that the Monitor observes on the local segment, but are destined for, or originated by, stations not on the local segment. All off-segment packets, then, are either routed to, from, or across the local segment.

Off-segment values are a measure of the amount of routing or bridging that is occurring. Excessive off-segment traffic may mean that certain stations on one segment are communicating primarily with stations on other segments. If you identify these stations and

move then to the segments where their primary communications partners are, you may lessen the overall loading on your network.

Off-segment Transit Packets

This parameter applies only to segment, not to nodes. The number of IP packets observed on the selected segment not destined for or originated by an object on the selected segment. For these packets, the selected segment is an intermediate hop in a route between the originating and destination segments.

Off-segment Transit Packets Rate

This parameter applies only to segments, not to nodes. The number of off-segment IP packets observed per second on the selected segment, not into or out of an object on the selected segment. For these packets, the selected segment is an intermediate hop in a route between the originating and destination segments.

ICMP Variables

ICMP Packets

ICMP (Internet Control Message Protocol) packets are used to control, test, and report problems with, the network. Reading through the ICMP variable descriptions should give you a good idea of how ICMP is used. A high number of ICMP packets from any source wastes traffic capacity that could otherwise be used for data packets.

Bytes

The Monitor maintains a count and rate for the number of ICMP bytes in and out of all monitored objects. A high number of ICMP bytes from any source wastes traffic capacity that could otherwise be used for data.

ICMP Errors

An ICMP error is logged when the Monitor observes an ICMP packet with an error in its ICMP header. For example, a packet may have a length field with an illegal value in it. A node that generates ICMP errors may be having software problems.

Off-segment

Off-segment packets are packets that the Monitor observes on the local segment that are destined for or sent by nodes not on the local segment. All off-segment packets are either routed to, from, or across the local segment.

A high number of ICMP packets from any source wastes traffic capacity that could otherwise be used for data packets. If there are a high number of in or transit off-segment ICMP packets, the source is on a different segment.

Destination Unreachable Packets

If for some reason a gateway cannot deliver an IP packet, it sends and ICMP Destination Unreachable packet to the sender. This packet informs the sender that the packet could not be delivered, and gives a reason. The Monitor keeps count of ICMP Destination Unreachable packets into and out of all objects, by reason. These are listed below.

Net unreachable

The network is having routing problems. Possible routing problems include: a non-operational link a node or router has an incorrect routing table

Host unreachable

See net unreachable.

Protocol unreachable

Port unreachable

Frag needed / DF set

This means fragmentation is needed but Don't Fragment flag was set. This message is sent when a router cannot forward a packet because it is too large for the next subnetwork in the route. Find out why fragmentation is being disallowed by the sending node - it may not be necessary. If it is necessary, then you must find or create an alternate route.

Source route failed

Destination net unknown

The destination network is not in the router's current routing table. This may be because the source node entered the address incorrectly (a software problem) or because the router's routing table is corrupt or incomplete.

Destination host unknown

See destination net unknown

Source host isolated

Destination net prohibited (communication with destination network administratively prohibited)

Net unreachable / TOS

This means network is unreachable for this Type of Service. This message is sent when a router cannot forward a packet because the specified Type of Service is not available for this route. Find out why this Type of Service is being specified. It may be unnecessary.

Host unreachable / TOS

This means host is unreachable for this Type of Service.

Time to Live Exceeded Packets

An IP packet is allowed to remain in transit for a fixed time. This time is called "time to live" and is specified in the IP packet by the sender. If this time expires before the packet is delivered, the packet is discarded. This mechanism prevents packets that get "stuck" in circular routes from congesting the network forever.

This mechanism is enforced by the gateways that route the packet through the network. Each gateway reduces the packet's timer value by an appropriate amount, and then checks to make sure that it has not reached zero. If the timer has reached zero, the gateway discards the packet and transmits an ICMP Time to Live Count Exceeded packet back to the sender.

Packets may get stuck in loops (circular routes) because a gateway or router has incorrect information in its routing table (example).

Reassembly Time Exceeded Packets

In routing an IP packet across a network, it is sometimes necessary to fragment it into smaller packets. This must be done to get it across a segment that cannot handle the packet at its original size.

Once a packet has been fragmented, it is not reassembled until the fragments reach the final destination. Since it is possible that one or more fragments will be lost before reaching the destination, the destination node waits only a fixed period of time to receive all the fragments. This is the reassembly time.

If the destination node has not received all of the fragments when the reassembly time expires, it sends an ICMP Fragment Reassembly Time Exceeded packet to the sender.

This problem typically occurs because one or more of the fragments has been lost.

Parameter Problem Packets

Part of each IP packet (the header) contains control information. A parameter is a unit of control information. For example, one parameter specifies the length of the packet, and another specifies whether or not fragmentation of this packet is allowed.

If a gateway detects a serious problem with a parameter, and it is not reportable through one of the other ICMP messages (such as Destination Unreachable), it sends an ICMP Parameter Problem packet back to the sender.

There is currently one specific reason tracked for the ICMP Parameter Problem packet:

Param option missing (missing option parameter)

Source Quench Packets

Gateways use the source quench mechanism to slow the rate of incoming packets. If a gateway is receiving packets too fast for it to keep up with, it will send

an ICMP Source Quench Packet to one or more nodes to tell them to slow down.

Redirect Packets

The redirect mechanism allows gateways to send information about routes to hosts. This works as follows:

Each node maintains a table that contains, for each of the nodes with which it communicates, the physical address of a gateway. This gateway is the first step in the route to the destination node. When a node sends a datagram to a node that is not on its segment, it send it to the gateway indicating in its routing table for the destination node.

Gateways maintain more or less complete routing information. They check all datagrams to be routed off a segment to make sure that the optimum route is being used. For example, if there are two gateways available to Node a, and Node A attempts to send a datagram to Node B across Gateway 1 when Gateway 2 would be better, Gateway 1 will detect the problem.

When this occurs, the detecting gateway issues an ICMP Redirect packet to the sending node. This packet tells the node how it should change its routing table.

Nodes use this mechanism to learn routes from gateways. All a node really needs on startup is to know the address of a gateway. It attempts to route all of its off-segment messages through this gateway, and builds its routing table from the ICMP Redirect packets it receives back.

An ICMP Redirect packet contains a diagnostic code that specifies additional information. The Monitor counts the occurrences of each of these:

Redirect for net

This packet means that datagrams to nodes on this network should be routed differently.

Redirect for host

This packet means that a datagram to this host should be routed differently.

Redirect to TOS net

This is a redirect for the network and type of service. This packet means that datagrams to hosts on this network should be routed differently in order to obtain this type of service.

Redirect TOS host

This is a redirect for the host and type of service. This packet means that a datagram to this host should be routed differently in order to obtain this type of service.

Echo Packets

The echo mechanism is used to verify that a destination is currently reachable, or to test the delay time between nodes. Echo is often referred to as "ping." The echo mechanism involves two ICMP packets: Echo Request and Echo Reply. The Monitor maintains counts for both of these.

Note that some diagnostic tools issue a series of ICMP Echo Request packets and then monitor and analyze the ICMP Echo Response packets.

A high number of these packets wastes traffic capacity.

Echo Request

This is a request that the addressed node send back an Echo Response packet.

Echo Response

This is a response packet sent by a node when it has received an Echo Request packet.

Timestamp Packets

The timestamp mechanism is used by nodes to synchronize their clocks. Node A sends an ICMP Timestamp Request packet to Node B, requesting that Node B return the current time of its system clock. Node B sends an ICMP Timestamp Response packet with the requested time to Node A. Node A can roughly synchronize its clock with Node B based on the response timestamp.

Timestamp Request

This is a request that the addressed node send back a Timestamp Response packet.

Timestamp Response

This is a response packet sent by a node when it has received a Timestamp Request packet.

Address Mask Packets

The IP protocol's addressing scheme allows sites to group multiple physical networks (segments) into a single addressable subnet. The subnet addressing scheme allows a site to determine, to an extent, which IP address bits identify the network (including subnet) and which identify nodes in the local subnet. For example, a site may determine that the first three octets in the IP address specify the network, and the last octet specifies the node in the network.

The division of address bits between network and node is represented by an address mask. The address mask is a string of 32 bits, where each bit used to specify network is set to 1, and bits that identify node are set to 0.

A node learns the address mask for its local subnet by requesting the information from a gateway. To do so it sends an ICMP Address Mask Request message to the gateway. If it does not know the address of the gateway, it may broadcast the request. The gateway replies with an ICMP Address Mask Response.

Address Mask Request

This is a request that the addressed node send back an Address Mask Response packet.

Address Mask Response

This is a response packet sent by a node when it has received an Address Mask Request packet.

<u>TCP Variables</u>

TCP Packets

A TCP packet (sometimes referred to as a segment) is a string of bytes that is transferred as a unit across

the IP network. It has two parts: the TCP header, which contains control information such as the source and destination TCP ports; and the data to be transferred to the destination user.

Bytes

The Monitor maintains a count and rate for bytes into and out of all monitored objects. For example, you can monitor the number of bytes into or out of a chosen node to measure the traffic load with respect to that node. You can monitor the number of bytes into and out of all nodes on the segment. The byte count includes header and data bytes.

Header Bytes

The header is the portion of the TCP packet that contains control information used by the protocol, such as source and destination TCP ports. Comparing the number of TCP header bytes to the total number of TCP bytes gives an idea of the amount of TCP overhead on a connection.

Error Packets

A TCP error is logged for each packet observed with one of the following problems:

* length of TCP packet is less than 20 bytes
* TCP Header length is less than 20 bytes
* TCP header length is greater than the length of the TCP packet
* TCP header length is greater than 20 bytes but the length of the TCP packet is less than the TCP header length.

Retransmissions

A Retransmission is a TCP packet that contains some data that has already been sent at least once. A Retransmission may or may not be an exact duplicate of the packet already transmitted.

Note that if the underlying packet delivery system (DLL) creates a duplicate, it is counted as a retransmission.

When a TCP entity sends a data packet to its remote partner, it waits a predetermined period of time (tracked by a retransmission timer) for an acknowledgement (ACK) from the remote partner. If this

time expires without the ACK being received, it retransmits the data contained in the presumably lost packet. It may retransmit a packet identical to the one lost, or it may combine data from multiple lost packets into a new packet, or it may combine lost data with new data into a new packet.

Excessive retransmissions can mean that a gateway is overloaded or down, that a system is overloaded, or that network parameters are misconfigured. In general, small dedicated networks should see few retransmissions. Larger, more diverse networks with routers, bridges and gateways with different capabilities and capacities are likely to have more retransmissions.

Bytes Retransmitted

Byte Retransmitted are TCP data bytes that have already been sent at least once.

See Retransmissions.

Out of Order Packets

Out of Order Packets are packets containing bytes with lower sequence numbers than bytes in previously seen packets.

Packets do not necessarily arrive in the order they were sent in. The receiving node puts the data in the correct order once it has received all packets. A high value may mean that some packets are being sent by way of a slower route, or that there is an overloaded or down bridge or router.

Out of Order Bytes

Out of Order Bytes are bytes with lower sequence numbers than bytes seen in previous packets.

Data out of Window Packets

Data out of Window Packets are packets that contains data that is not within the boundaries of the receiving partner's currently advertised window. The data is either acknowledged data or data that the partner is not ready to receive.

Bytes out of Window

Bytes out of Window are bytes that are not within the boundaries of the receiving partner's currently advertised window. The data is either acknowledged data or data that the partner is not ready to receive.

Packets after Close

Packets after Close are packets observed after a connection has been closed. These may be packets that had been "lost" on the network, or it may indicate a malfunction in the sending station.

RST Packets

A packet in which the RST (reset) bit is set.

SYN Control Packets

A packet in which the SYN bit is set.

FIN Control Packets

A packet in which the FIN bit is set.

URG Control Packets

An URG Control Packet is a packet in which the Urgent pointer is set.

The packet contains data that the receiving application should process as soon as possible. For example, the control-key sequences used by some applications are often sent as Urgent data.

Keepalives

A Keepalive is a TCP packet that a user sends to check to see if a connection is still active. The Keepalive packet contains either not data or one garbage byte of data that is outside the remote partner's last advertised window. The remote partner responds with either an ACK, confirming that the connection is alive, or a RST, indicating that the connection had been dropped.

Although widely implemented, the keepalive mechanism is not part of the TCP protocol, so you will not necessarily see keepalive activity.

Keepalives mean that a connection has been up for a long time without and activity. Resources may be unnecessarily tied up.

Window Probes

A Window Probe is a TCP packet that is sent to check the size of the remote partner's window when the last advertised window size was zero. The Window Probe packet contains one byte of data. The remote partner responds with an ACK packet, which contains the size of the remote partner's current window size.

Non-data packets, which may include window update information, may be lost and are not be retransmitted. It may therefore become necessary to check the remote partner's window size if that information has not been received for some period of time. This can mean that a node is runnind a faulty TCP implementation, that timers are misconfigured, or packets are being lost.

Window Update Only Packets

A Window Update Only packet is a packet that contains no data, but in which the advertised window size has been updated.

APPENDIX IV

Summary Tool - Values Display Fields

Packet Rate	total packets per second at this protocol layer received and transmitted at segment or node
Byte Rate	total bytes per second at this protocol layer received and transmitted at segment or node
Errors	total errors at this protocol layer received and transmitted at segment or node
Broadcast Pkt Rate	total number packets per second at this protocol layer addressed to broadcast address
Multicast Pkt Rate	total number packets per second at this protocol layer addressed to multicast address
Source Quenches	total number of ICMP source quench packets received and transmitted from this segment or node.
Fragments	total number of IP fragmented packets received and transmitted from this segment or node.
Flow Controls	
UDP	total number of ICMP source quench packets received and transmitted on this UDP port.
TCP	total number of ICMP source quench packets received and transmitted on this TCP port.
NFS	total number of ICMP source quench packets received and transmitted on this NFS port.
Retransmissions	total number of TCP packets retransmitted on this TCP port.
Off Segment Packets	
in	%traffic at this protocol layer received by nodes on this segment originating from other segments
	in = 100(packet rate / packet rate rcv from off seg)
out	% traffic at this protocol layer transmitted by nodes on this segment to nodes on other segments
	out = 100(packet rate / packet rate xmt to off seg)
Transit	% traffic at this protocol layer originating from other segments which are addressed to nodes not on this segment
	transit = 100(packet rate / packet rate transit)
Local	% Traffic at this protocol layer which originates and terminates on this segment
	local = 100 -(in + out + transit)
Most Active Protocols	The five most active protocols running above this layer (ie the users of this layer). The protocols are displayed as % and ranked in decreasing order.
	protocol % = 100(protocol packet rate/packet rate)

Most Active Nodes	The five most active nodes at this protocol layer . The nodes are displayed as % and ranked in decreasing order. node % = 100(node packet rate/packet rate)
ICMP Types Seen	The total number of these specific ICMP packet types transmitted and received on this segment or node.
Total Segment Bandwidth	The % of the available bandwidth used by this protocol. If the screen is a segment display it is % used by all nodes on the segment, if it is a node display it is the % used by that node. % = 100(8 * frame rate / 10000000)
Total Active Dialogs	The number of dialogs detected for the node or segment at this protocol layer.

5. Actual Screens for Values Tool

5.1 Data Link Group

5.1.1 Definition

This screen summarizes the data link parameters.

5.1.2 Defaults

1 This is a "complete values" screen. It shows all of the values for the DLL protocol layer.

2 The user comes from a context of a specific segment or node and this screen must preserve that context.

5.1.3 Primary Screen Layout

Standard Column Headings
Frames
Rcv
Xmt
Total
Frm rate
Rcv
Xmt
Total
Bytes
Rcv
Xmt
Total
Byte rate
Rcv
Xmt
Total
Errors
Rcv
Xmt
Total
Error rate
Rcv
Xmt
Total
802.3 frames
Rcv
Xmt
Total
ethernet frames
Rcv
Xmt
Total
802.3 frame rate
Rcv
Xmt
Total
ethernet frame rate
Rcv
Xmt
Total
Bcast Xmt
Bcast rate
Mcast Xmt
Mcast rate
Off seg
Rcv
Xmt
[Transit]

[local]
Total
Off seg rate
Rcv
Xmt
[Transit]
[local]
Total
Runts Xmt
[Allignment]
[Collisions]

Protocol	Pkt Count	Pkt Rate	%
Protocol 1			
Protocol 2			
.			
.			
Protocol n			

5.1.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screen

5.2 IP Group

5.2.1 Definition

This screen provides information for the IP network layer running on the segment or node.

5.2.2 Defaults

1 This is a "complete values' screen. It shows all of the values for the IP protocol type

2 The user comes from a context of a specific segment or node and this screen must preserve that context

5.2.3 Primary Screen Layout

Standard Column Headings
Pkts
Pkt rate
Bytes
Byte rate
Errors
Error rate
Frags
Frag rate
Header bytes
Header rate
Bcast Xmt
Bcast rate
Mcast Xmt
Mcast rate
Off seg
Off seg rate

Protocol	Pkt Count	Pkt Rate	%
Protocol 1			
Protocol 2			
.			
.			
Protocol n			

5.2.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screen

5.3 ICMP Group

5.3.1 Definition

This screen provides information for the ICMP protocol s/w running on the segment or node.

5.3.2 Defaults

1 This is a "complete values" screen. It shows all of the values for the ICMP protocol type

2 The user comes from a context of a specific segment or node and this screen must preserve that context.

5.3.3 Primary Screen Layout

Standard Column Headings
Pkts
Pkt rate
Bytes
Byte rate
Errors
Error rate
Off seg
Off seg rate
D.U. net
D.U. host
D.U. Prot
D.U. port
D.U. frag
D.U. Src route
D.U. Net Unk.
D.U. Host Unk.
D.U. Src Host isol.
D.U. Dnet Ad Prob
D.U. Dhost Ad Prob
D.U. Net Unr.
D.U. Time Xd Trans
D.U. Time Xd Reass
Param prob
Param opt miss.
src quench
redir net
redir host
redir tos net
redir tos host
Echo req
Echo Resp
Ts req
Ts resp
Addr mask req
Addr mask resp

5.3.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screen

5.4 UDP Group

5.4.1 Definition

This screen provides information for the UDP protocol s/w running on the segment or node.

5.4.2 Defaults

1 This is a "complete values" screen. It shows all of the values for the UDP protocol type

2 The user comes from a context of a specific segment or node and this screen must preserve that context.

5.4.3 Primary Screen Layout

	Standard Column Headings
Pkts	
Pkt rate	
Bytes	
Byte rate	
Errors	
Error rate	
Header bytes	
Header rate	
off seg	
off seg rate	

Protocol	Pkt Count	Pkt Rate	%
Protocol 1			
Protocol 2			
.			
.			
Protocol n			

5.4.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screen

5.5 TCP Group

5.5.1 Definition

This screen provides information for the TCP protocol s/w running on the segment or node.

5.5.2 Defaults

1 This is a "complete values" screen. It shows all of the values for the TCP protocol type

2 The user comes from a context of a specific segment or node and this screen must preserve that context

5.5.3 Primary Screen Layout

Standard Column Headings
number connections
Pkts
Pkt rate
bytes
Byte rate
header bytes
Hdr byt rt
errors
Error rate
persists
keep alives
rexmits
bytes rexmit
ack only pkt
window probes
pkts urg only
window update only
control pkts
dup only pkts
part dup pkts
dup bytes
out order pkts
out order bytes
data pkts after window
bytes after window
pkts after close
dup acks
ack pkts
off seg
off seg rate

Protocol	Pkt Count	Pkt Rate	%
Protocol 1			
Protocol 2			
.			
.			
Protocol n			

5.5.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screens

5.6 NFS Group

5.6.1 Definition

These screens provide information for the NFS protocol s/w running on the segment or node. The screens show the breakdown of activity by servers and clients for filesystems, directories and files.

5.6.2 Defaults -client/server

1 This is a "complete values" screen. It shows all of the values for the NFS protocol type

2 The user comes from a context of either a segment or a node and this screen must preserve that context.

5.6.3 Primary Screen Layout -client/server

Standard Column Headings
total nfs ops
nfs ops rate
read opss
read rate
write ops
bytes read
bte read rate
bytes written
bytes written rate
write rate
write cache
create file
remove file
rename file
create dir
remove dir
null ops
get file attr
set file attr
look ups
read link
create link
create sym lnk
get fsys attr
mount
unmount
readmount
unmountall
readexport

File Systems on Server
file system 1
file system 2
.
.
file system n

5.6.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screens

5.6.5 Navigation

Double clicking on a file system invokes the file system screen for the selected file system.

5.6.6 Defaults -file system

1 This is a "complete values" screen. It shows all of the values for the NFS protocol type for this file system.

2 The user comes from a context of either an nfs client or server and this screen must preserve that context.

5.6.7 Primary Screen Layout -file system

Standard Column Headings
total nfs ops
nfs ops rate
read ops
read op rate
write ops
write op rate
bytes read
bte read rate
bytes written
bytes written rate
write cache
create file
remove file
rename file
create dir
remove dir
null ops
get file attr
set file attr
look ups
read link
create link
create sym lnk
get fsys attr
mount
unmount

Directories in File System
directory 1
directory 2
.
.
directory n

5.6.8 Secondary Screen Layout

Extended Column Headings
rows as for primary screens

5.6.9 Navigation

Double clicking on a directory invokes the directory screen for the selected directory.

5.6.10 Defaults -directory

1 This is a "complete values" screen. It shows all of the values for the NFS protocol type for this directory.

2 The user comes from a context of an nfs file system and this screen must preserve that context.

5.6.11 Primary Screen Layout -directory

Standard Column Headings
total nfs ops
nfs ops rate
read ops
read ops rate
write ops
write ops rate
bytes read
bte read rate
bytes written
bytes written rate
write cache
create file
remove file
rename file
null ops
get file attr
set file attr
look ups
read link
create link
create sym lnk
create sym lnk

Attributes
type
mode
nlinks
uid
gid
size
blocksize
rdev
blocks
fileid
atime
mtime
ctime

Files in Directory
file 1
file 2
.
.
file n

5.6.12 Secondary Screen Layout

Extended Column Headings
rows as for primary screens

5.6.13 Navigation

Double clicking on a file invokes the file screen for the selected file.

5.6.14 Defaults -file

1 This is a "complete values" screen. It shows all of the values for the NFS protocol type for this file.

2 The user comes from a context of an nfs file directory and this screen must preserve that context.

5.6.15 Primary Screen Layout -file

Standard Column Headings
total nfs ops
nfs ops rate
read ops
read ops rate
write ops
write ops rate
bytes read
bte read rate
bytes written
bytes written rate
write cache
null ops
get file attr
set file attr
look ups
read link
create link
create sym lnk
Attributes
type
mode
nlinks
uid
gid
size
blocksize
rdev
blocks
fileid
atime
mtime
ctime

5.6.16 Secondary Screen Layout

Extended Column Headings
rows as for primary screens

5.7 ARP Group

5.7.1 Definition

This screen provides information for the ARP protocol s/w running on the segment or node.

5.7.2 Defaults

1 This is a "complete values" screen. It shows all of the values for the ARP protocol type

2 The user comes from a context of either a segment or a node and this screen must preserve that context.

5.7.3 Primary Screen Layout

Standard Column Headings
TBD

5.7.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screens

5.8 RARP Group

5.8.1 Definition

This screen provides information for the RARP protocol s/w running on the segment or node.

5.8.2 Defaults

1 This is a "complete values" screen. It shows all of the values for the RARP protocol type

2 The user comes from a context of either a segment or a node and this screen must preserve that context.

5.8.3 Primary Screen Layout

Standard Column Headings
TBD

5.8.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screens

5.9 Telnet Group

5.9.1 Definition

This screen provides information for the Telnet protocol s/w running on the segment or node.

5.9.2 Defaults

1 This is a "complete values" screen. It shows all of the values for the Telnet protocol type

2 The user comes from a context of either a segment or a node and this screen must preserve that context.

5.9.3 Primary Screen Layout

Standard Column Headings
TBD

5.9.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screens

5.10 FTP Group

5.10.1 Definition

This screen provides information for the FTP protocol s/w running on the segment or node.

5.10.2 Defaults

1 This is a "complete values" screen. It shows all of the values for the FTP protocol type

2 The user comes from a context of either a segment or a node and this screen must preserve that context.

5.10.3 Primary Screen Layout

Standard Column Headings
TBD

5.10.4 Secondary Screen Layout

Extended Column Headings
rows as for primary screens

5.11 Dialogue Data Group

5.11.1 Definition

This screen displays all of the Data available for a particular dialogue. This screen is shown when the user clicks on an entry in the Summary Tool dialogue information.

Each dialog screen represents a single dialog. Thus at the UDP or TCP level two nodes may have multiple dialogs (each with a unique port pair) and each of these will be represented as a seperate entity.

Because the user cannot uniquely identify the dialog he requires from the menus (he does not know the port numbers involved) the only mechanism to invoke these screens is by selection of a dialog from the appropriate summary screen. This problem also prevents the user from 'clicking' through all the dialogs on ports between a node pair (may be addressed in later phase).

5.11.2 Defaults

1 This is a "complete values" screen. It shows all of the values available for the selected connection.

2 There are several different contexts for this screen. The user may select this option from the summary tools for all protocols. This screen must reflect the node, layer and specific connection context from which the user entered

The content of this screen is essentially the same as the corresponding row entry from the Traffic matrix screen for the DLL and IP layers. Their inclusion is to provide the user with a consistent navigaion paradigm accross the layers (and to provide this functionality in release 1 which does ot include the Traffic matrix support).

The data set displayed in this screen will be appropriate to the protocols used between the nodes. The variables shown are those selected for TCP/IP protocols. Where nodes converse using multiple protocols this will be expanded to select data from each protocl set.

5.11.3 Primary Screen -DLL

node name	node name
mac address	mac address
ip address	ip address
Network Protocols:	
start time	last seen time

Standard Column Headings

frames
bytes
errors
flow ctl
ip frags
tcp retransmissions

5.11.4 Secondary Screen Layout -DLL

Extended Column Headings

rows as for primary screens

5.11.5 Primary Screen -IP

node name	node name
mac address	mac address
ip address	ip address
Transport Protocols:	
start time	last seen time

Standard Column Headings

Pkts
bytes
header bytes
errors
fragments
TCP retransmissions
ICMP

5.11.6 Secondary Screen Layout -IP

Extended Column Headings

rows as for primary screens

5.11.7 Primary Screen -ICMP

This is invoked by selection of the ICMP entry from the IP screen.

node name mac address ip address	node name mac address ip address

Standard Column Headings

Pkts
Bytes
Errors
Off seg
D.U. net
D.U. host
D.U. Prot
D.U. port
D.U. frag
D.U. Src route
D.U. Net Unk.
D.U. Host Unk.
D.U. Src Host isol.
D.U. Dnet Ad Prob
D.U. Dhost Ad Prob
D.U. Net Unr.
D.U. Time Xd Trans
D.U. Time Xd Reass
Param prob
Param opt miss.
src quench
redir net
redir host
redir tos net
redir tos host
Echo req
Echo Resp
Ts req
Ts resp
Addr mask req
Addr mask resp

5.11.8 Secondary Screen Layout

Extended Column Headings

rows as for primary screens

5.11.9 Primary Screen -UDP

node name	node name
mac address	mac address
ip address	ip address
port number	port number
Application Protocol:	
start time	last seen time

Standard Column Headings

Pkts
bytes
errors
ip frags
flow ctl

5.11.10 Secondary Screen Layout -UDP

Extended Column Headings

rows as for primary screens

5.11.11 Primary Screen -TCP

node name	node name
mac address	mac address
ip address	ip address
port number	port number
Application Protocol:	
Connection Status: [active, closed-ok, closed reset, unknown]	
start time	last seen time

Standard Column Headings
Pkts
bytes
header bytes
errors
pkts bad seq #
bytes not acked
persists
keep alives
pkts rexmit
bytes rexmit
ack only pkt
window probes
pkts urg only
window update only
control pkts
dup only pkts
part dup pkts
dup bytes
out order pkts
out order bytes
data pkts after window
bytes after window
pkts after close
dup acks
acks unsent data
ack pkts
bytes acked by acks
current window

5.11.12 Secondary Screen Layout -TCP

Extended Column Headings

rows as for primary screens

5.11.13 Primary Screen -NFS

node name	node name
mac address	mac address
ip address	ip address
port number	port number
start time	last seen time

Standard Column Headings

variables as for NFS Group

5.11.14 Secondary Screen Layout -NFS

Extended Column Headings

rows as for primary screens

5.11.15 Navigation

As for NFS group a hieararchy of screens is available:

1 client to server

2 client to file system

3 client to directory

4 client to file

5.12 Traffic Matrix Group (Not in release 1)

5.12.1 Definition

This screen shows traffic distribution between a selected node (or segment) and other nodes (or segments) in the network.

For the DLL and IP layers it is essentially a repeat of the dialogue screens. For the UDP and TCP layers however it represents a summation over multiple connections between the two nodes.

5.12.2 Defaults

1 The user comes from a context of a specific segment or node plus a protocol level and this screen must preserve this context.

2 If the selection propagated from the Summary Tool is a segment then the distribution is segment to segment. If the selection is a node then the distribution is node to node.

3 Values are shown in order of heaviest traffic to lightest.

4 The initial screen has the heaviest pairs of nodes or segments. Scrolled screens contain progressively lighter traffic loads.

5 The user can select the column by which the nodes are to be ordered and request reordering. This allows the user to use this screen look at flow control for example.

6 Double clicking on a node or segment in the display area allows the user to move to this object as the focus of the traffic matrix ie if the user is looking at a matrix for node A and selects node B (which is one of the nodes in the matrix) they will get the traffic matrix for B.

7 Double clicking on the node which is the focus of the matrix (eg A in the above example) selects the next segment or node, consistent with the current view. Node views click to other nodes on the segment. Segment views click to other segments. The segment (or) node selection will be ordered alphabetically.

8 The data maintained between two nodes (or segments) will be aged out if no communication between them occurs for a defined period (settable by the user -eventually).

5.12.3 Primary Screen DLL

Node(Segment) Name

	frm	frm rate	byte	byte rate	err	err rate	flow ctl	flow ct rt	tffc %
node(segment)1									
node(segment)2									
.									
.									
.									
node(segment)n									

This scrolls down to accomodate all nodes (or segments) required.

5.12.4 Secondary Screen

	frag	frag rate	tcp rexmit	rexm rt	tcp

rows as primary screen

5.12.5 Primary Screen IP

Node(Segment) Name

	pkt	pkt rate	err	err rate	frag	frag rate	icmp	flw ctl	flw ct rt	tffc %
node(segment)1										
node(segment)2										
.										
.										
.										
node(segment)n										

This scrolls down to accomodate all nodes (or segments) required.

5.12.6 Primary Screen ICMP

This is invoked by selection of the ICMP entry for a node (segment) pair. The user is vectored to the IP traffic matrix screen in this case.

5.12.7 Primary Screen TCP

Node(Segment) Name

	pkt	pkt rate	err	err rate	act conn	rxmt	rxmt rate	flw ctl	flw ct rt	tffc %	# conns
node(segment)1											
node(segment)2											
.											
.											
.											
node(segment)n											

This scrolls down to accomodate all nodes (or segments) required.

5.12.8 Primary Screen UDP

Node(Segment) Name

	pkt	pkt rate	err	err rate	actv conn	flow ctl	flow ctl rt	%	tffc
node(segment)1									
node(segment)2									
.									
.									
.									
node(segment)n									

This scrolls down to accomodate all nodes (or segments) required.

5.12.9 Primary Screens NFS

5.12.9.1 Client to Server

Node(Segment) Name

	pkt	pkt rate	err	err rate	actv conn	flow ctl	flow ctl rt	tffc %
node(segment)1								
node(segment)2								
.								
.								
.								
node(segment)n								

File systems on this node

file system 1
file system 2
.
.
file system n

This scrolls down as required.

5.12.9.1.1 Navigation

Double clicking on a file system invokes the file system screen for the selected file system.

5.12.9.2 *Client to File System*

Node(Segment) Name
File System name

	pkt	pkt rate	err	err rate	actv conn	flow ctl	flow ctl rt %	tffc
node(segment)1								
node(segment)2								
.								
.								
.								
node(segment)n								

Directories on this file system

directory 1
directory 2
.
.
directory n

This scrolls down as required.

5.12.9.2.1 Navigation

Double clicking on a directory invokes the directory screen for the selected directory.

5.12.9.3 *Client to Directory*

Node(Segment) Name
File System name
directory name

	pkt	pkt rate	err	err rate	actv conn	flow ctl	flow ctl rt %	tffc
node(segment)1								
node(segment)2								
.								
.								
.								
node(segment)n								

files in this directory

file 1
file 2
.
.
file n

This scrolls down as required.

5.12.9.3.1 Navigation

Double clicking on a file invokes the file screen for the selected file.

4.12.9.7 Client to File

Node(Segment) Name
File System name
directory name
file name

	pkt	pkt rate	err	err rate	actv conn	flow ctl	flow ctl rt %	tffc
node(segment)1								
node(segment)2								
.								
.								
.								
node(segment)n								

This scrolls down as required.

5.13 Summary Screen for Traffic Matrix

	Seg1	Seg2	Seg3	....	Segn
Seg1		frame byte error	frame byte error		frame byte error
Seg2	frame byte error		frame byte error		frame byte error
Seg3	frame byte error	frame byte error			frame byte error
.					
.					
.					
Segn	frame byte error	frame byte error	frame byte error	....	

Claims

1. A method for monitoring communications which occur in a network of nodes, each communication being effected by a transmission of one or more packets among two or more communicating nodes, each communication complying with a predefined communication protocol selected from among protocols available in said network, said method comprising

detecting passively and in real time the contents of packets, and

deriving, from said detected contents of said packets, communication information associated with multiple said protocols.

2. The method of claim 1 wherein said step of deriving communication information includes deriving communication information from associated with multiple layers of at least one of said protocols.

3. A method for monitoring communication dialogs which occur in a network of nodes, each dialog being effected by a transmission of one or more packets among two or more communicating nodes, each dialog complying with a predefined communication protocol selected from among protocols available in said network, said method comprising

detecting the contents of packets, and

deriving from said detected contents of said packets, information about the states of dialogs occurring in said network and which comply with different selected protocols available in said network.

4. The method of claim 3 wherein said step of deriving information about the states of dialogs comprises

maintaining a current state for each dialog, and

updating the current state in response to the detected contents of transmitted packets.

5. The method of claim 3 wherein said step of deriving information about the states of dialogs comprises

maintaining, for each dialog, a history of events based on information derived from the contents of packets, and

analyzing the history of events to derive information about the dialog.

6. The method of claim 5 wherein said step of analyzing the history includes counting events.

7. The method of claim 5 wherein said step of analyzing the history includes gathering statistics about events.

8. The method of claim 5 further comprising

monitoring the history of events for dialogs which are inactive, and

purging from the history of events dialogs which have been inactive for a predetermined period of time.

9. The method of claim 4 wherein said step of deriving information about the states of dialogs comprises

updating said current state in response to observing the transmission of at least two data related packets between nodes.

10. The method of claim 5 wherein said step of analyzing the history of events comprises

analyzing sequence numbers of data related packets stored in said history of events, and

detecting retransmissions based on said sequence numbers.

11. The method of claim 4 further comprising

updating the current state based on each new packet associated with said dialog, and

if an updated current state cannot be determined, consulting information about prior packets associated with said dialog as an aid in updating said state.

12. The method of claim 5 further comprising

searching said history of events to identify the initiator of a dialog.

13. The method of claim 5 further comprising

searching the history of events for packets which have been retransmitted.

14. The method of claim 4 wherein

the full set of packets associated with a dialog up to a point in time completely define a true state of the dialog at that point in time,

said step of updating the current state in response to the detected contents of transmitted packets comprises generating a current state which may not conform to the true state.

15. The method of claim 5 wherein the step of updating the current state comprises updating the current state to "unknown".

16. The method of claim 14 further comprising updating the current state to the true state based on

information about prior packets transmitted in the dialog.

17. The method of claim 15 further comprising updating the current state to the true state based on information about prior packets transmitted in the dialog.

18. The method of claim 3 wherein said step of deriving information about the states of dialogs occurring in said network comprises parsing said packets in accordance with more than one but fewer than all layers of a protocol.

19. The method of claim 3 wherein each said communication protocol includes multiple layers, and each dialog complies with one of said layers.

20. The method of claim 3 wherein said protocols include a connectionless-type protocol in which the state of a dialog is implicit in transmitted packets, and said step of deriving information about the states of dialogs includes inferring the states of said dialogs from said packets.

21. The method of claim 4 further comprising parsing said packets in accordance a protocol and temporarily suspending parsing of some layers of said protocol when parsing is not rapid enough to match the rate of packets to be parsed.

22. A method of analyzing the performance of a network of nodes which communicate via dialogs, each dialog being effected by a transmission of one or more packets among two or more communicating nodes, each

dialog complying with a predefined communication protocol selected from among protocols available in said network, said method comprising

monitoring the operation of the network with respect to specific items of performance during normal operation,

generating a model of said network based on said monitoring, and

setting acceptable threshold levels for said specific items of performance based on said model.

23. The method of claim 22 further comprising

monitoring the operation of the network with respect to the specific items of performance during periods which may include abnormal operation.

24. Apparatus for monitoring communication dialogs which occur in a network of nodes, each dialog being effected by a transmission of one or more packets among two or more communicating nodes, each dialog complying with a predefined communication protocol selected from among protocols available in said network, said apparatus comprising

a monitor connected to the network medium for passively, and in real time, monitoring transmitted packets and storing information about dialogs associated with said packets, and

a workstation for receiving said information about dialogs from said monitor and providing an interface to a user.

25. The apparatus of claim 24 wherein said workstation further comprises

means for enabling a user to observe events of acitve dialogs.

26. Apparatus for monitoring packet communications in a network of nodes in which communications may be in accordance with multiple protocols, said apparatus comprising

a monitor connected to a communication medium of the network for passively, and in real time, monitoring transmitted packets of different protocols and storing information about communications associated with said packtes, said communications being in accordance with different protocols, and

a workstation for receiving said information about said communciations from said monitor and providing an interface to a user,

said monitor and said workstation including means for relaying said information about multiple protocols with respect to communication in said different protocols from said monitor to said workstation in accordance with a single common network management protocol.

27. A method of diagnosing communication problems between two nodes in a network of nodes interconnected by links, comprising

monitoring the operation of the network with respect to specific items of performance during normal operation,

generating a model of normal operation of said network based on said monitoring, and

setting acceptable threshold levels for said specific items of performance based on said model.

28. The method of claim 27 further comprising the steps of

monitoring the operation of the network with respect to the specific items of performance during periods which may include abnormal operation, and

when abnormal operation of the network with respect to communication between the two nodes is detected, diagnosing the problem by separately analyzing the performance of each of the nodes and each of the links connecting the two nodes to isolate the abnormal operation.

29. A method of timing the duration of a transaction of interest occurring in the course of communication between nodes of a network, the beginning of said transaction being defined by the sending of a first packet of a particular kind from one node to the other, and the end of said transaction being defined by the sending of another packet of a particular kind between the nodes, comprising

passively and in real time monitoring packets transmitted in the network,

beginning to time said transaction upon the appearance of said first packet,

determining when the other packet has been transmitted, and

ending the timing of the duration of the transaction upon the appearance of the other packet.

30. A method for tracking node address to node name mappings in a network of nodes of the kind in which each node has a possibly nonunique node name and a unique node address within the network and in which node addresses can be assigned and reassigned to node names dynamically using a name binding protocol message incorporated within a packet, said method comprising

monitoring packets transmitted in said network, and

updating a table linking node names to node addresses based on information contained in said name binding protocol messages in said packets.

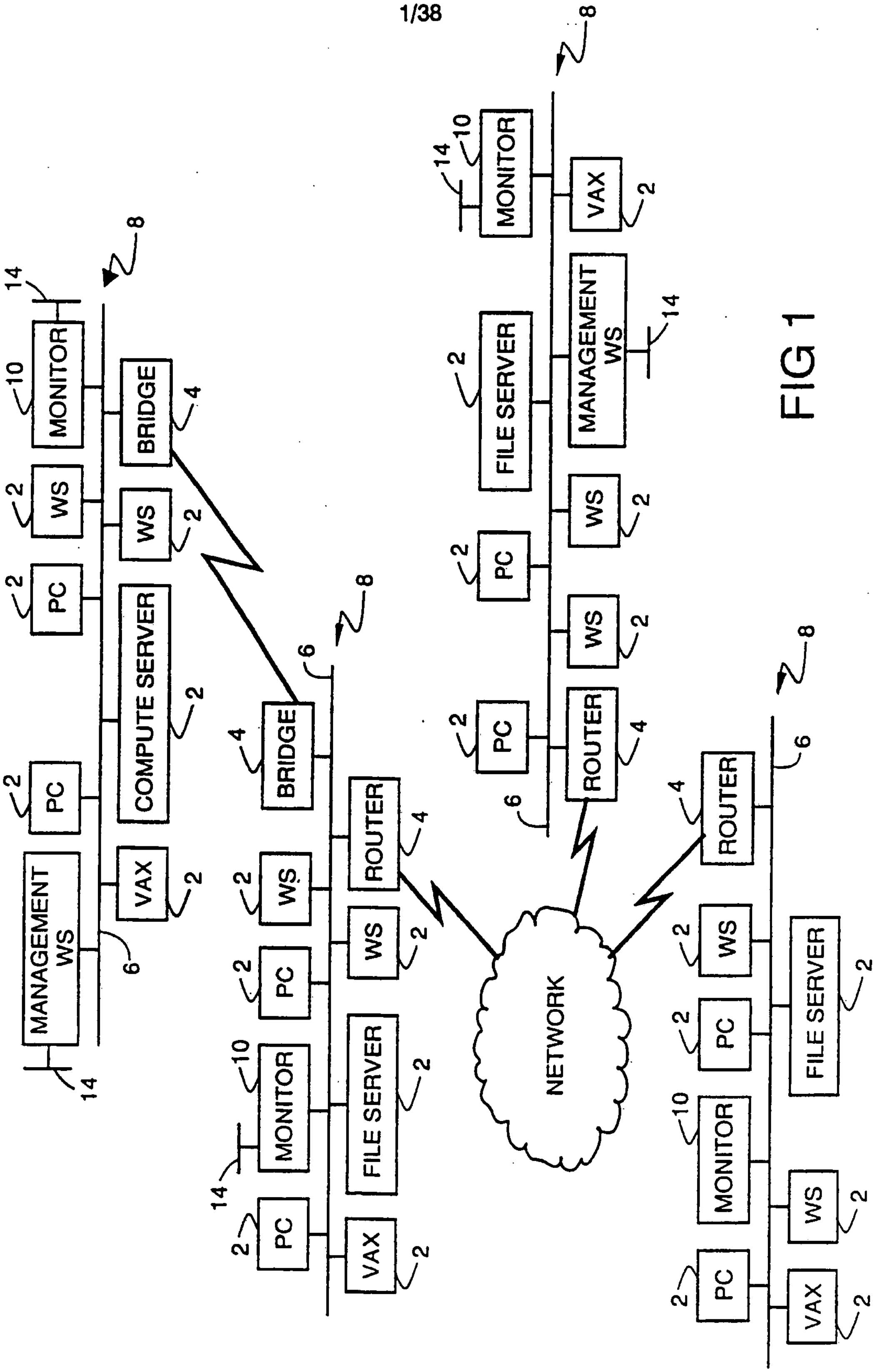

MONITOR
WS
PC
COMPUTE SERVER
BRIDGE
VAX
MANAGEMENT WS
ROUTER
FILE SERVER
NETWORK

FIG 1

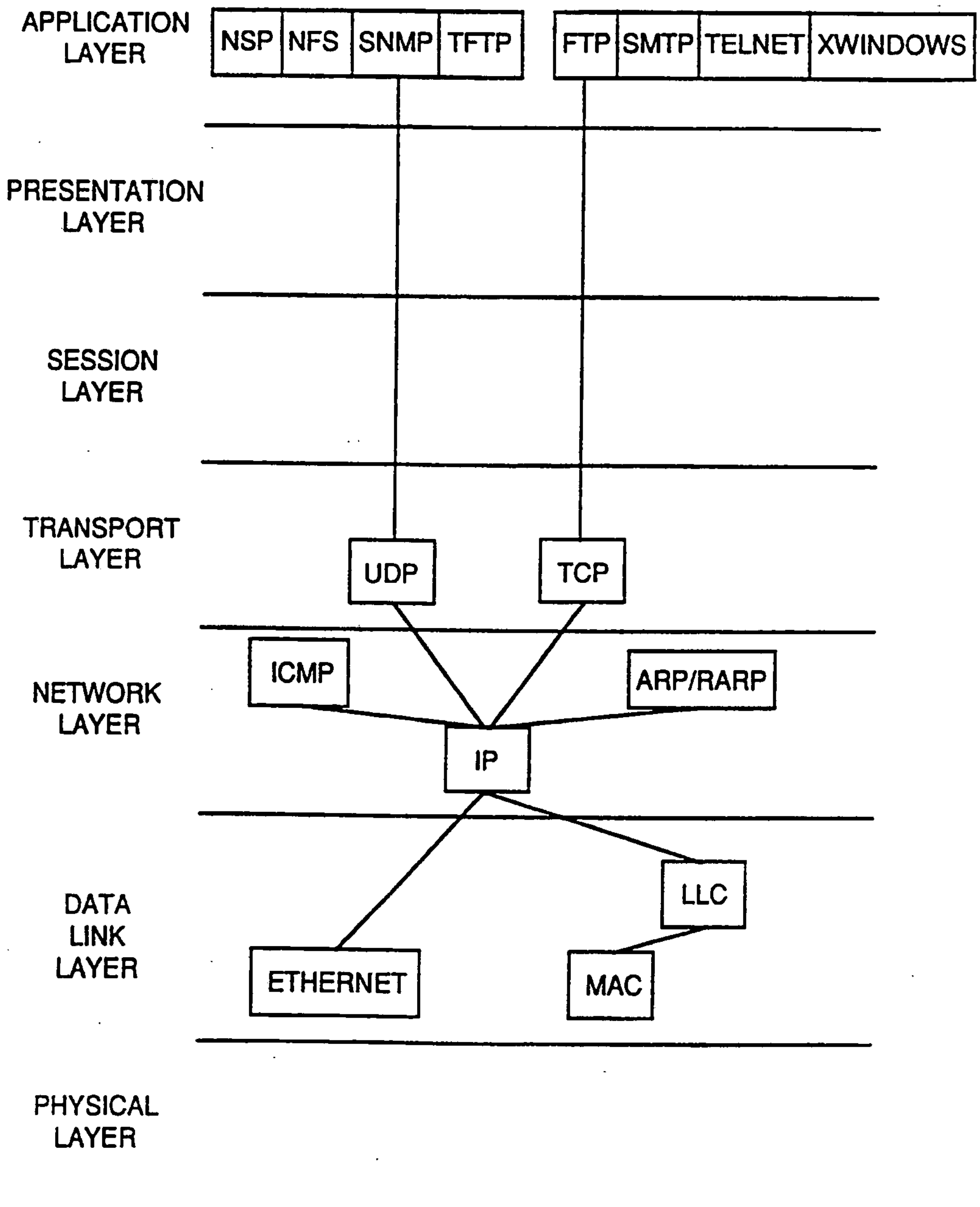

APPLICATION LAYER
NSP
NFS
SNMP
TFTP
FTP
SMTP
TELNET
XWINDOWS
PRESENTATION LAYER
SESSION LAYER
TRANSPORT LAYER
UDP
TCP
NETWORK LAYER
ICMP
ARP/RARP
IP
DATA LINK LAYER
LLC
ETHERNET
MAC
PHYSICAL LAYER

FIG 2

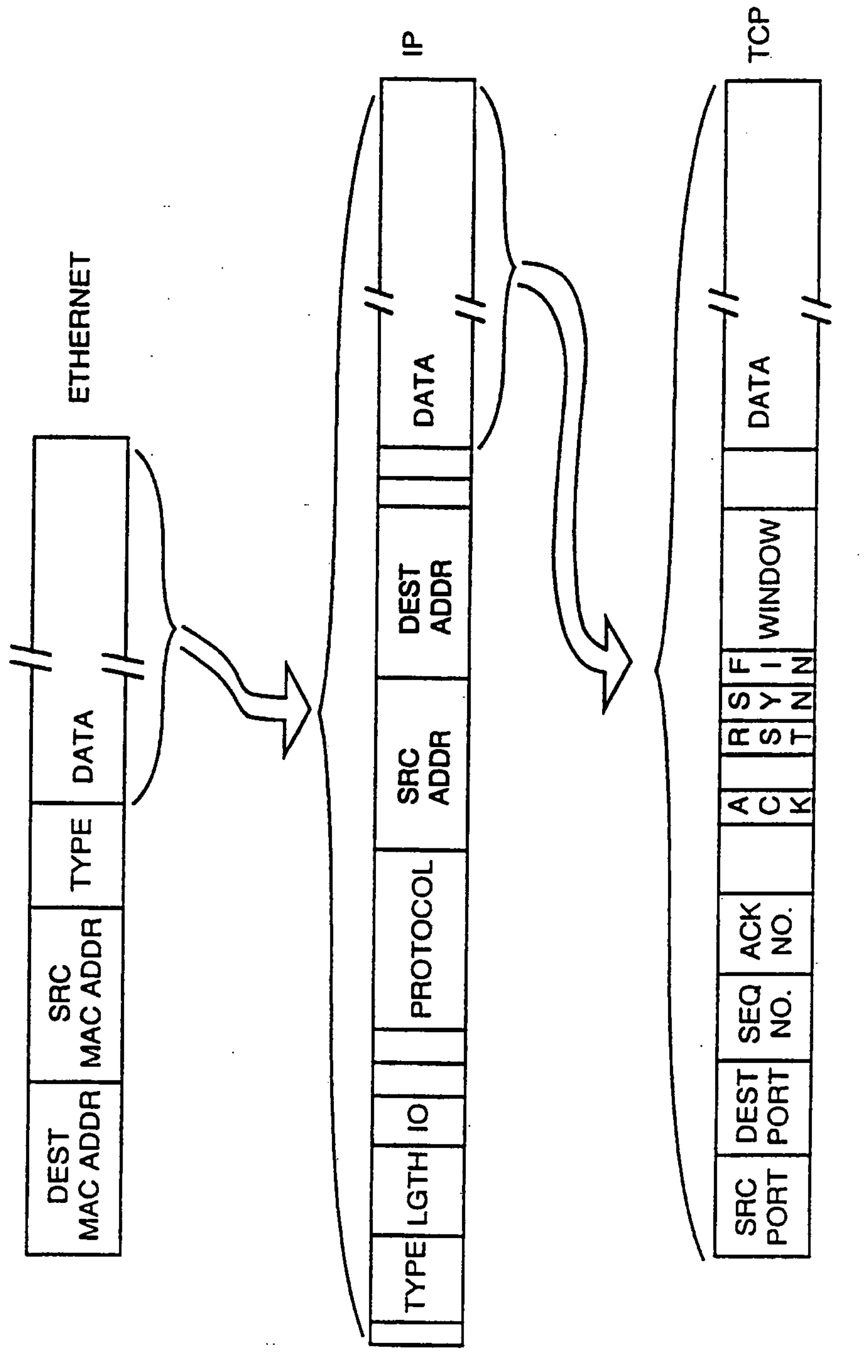

ETHERNET
DEST MAC ADDR
SRC MAC ADDR
TYPE
DATA
IP
TYPE
LGTH
IO
PROTOCOL
SRC ADDR
DEST ADDR
DATA
TCP
SRC PORT
DEST PORT
SEQ NO.
ACK NO.
ACK
RST
SYN
FIN
WINDOW
DATA

FIG 3

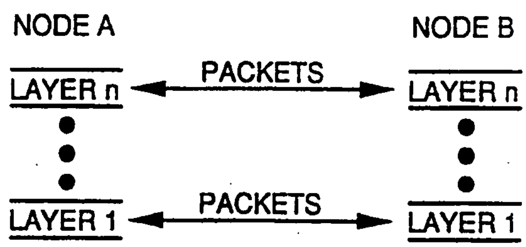


FIG 4

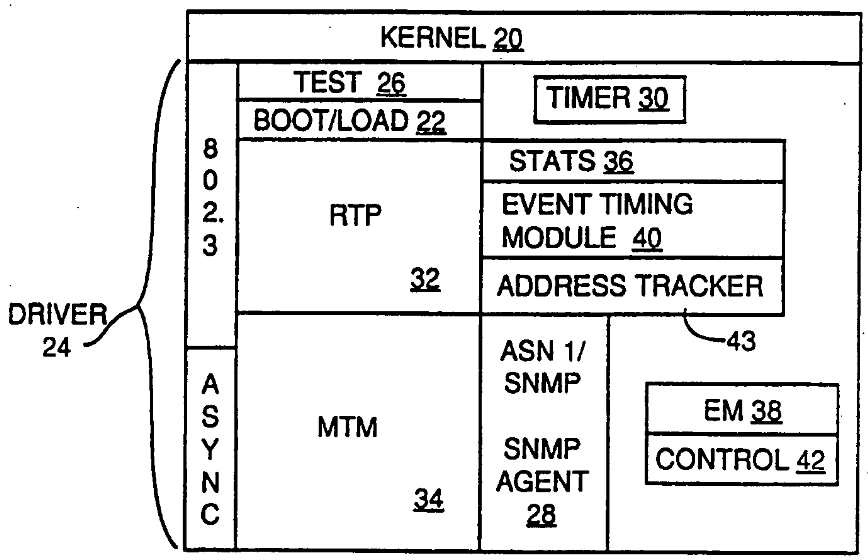

KERNEL 20
TEST 26
BOOT/LOAD 22
TIMER 30
8
0
2.
3
RTP
32
STATS 36
EVENT TIMING
MODULE 40
ADDRESS TRACKER
43
DRIVER
24
A
S
Y
N
C
MTM
34
ASN 1/
SNMP
SNMP
AGENT
28
EM 38
CONTROL 42

FIG 5

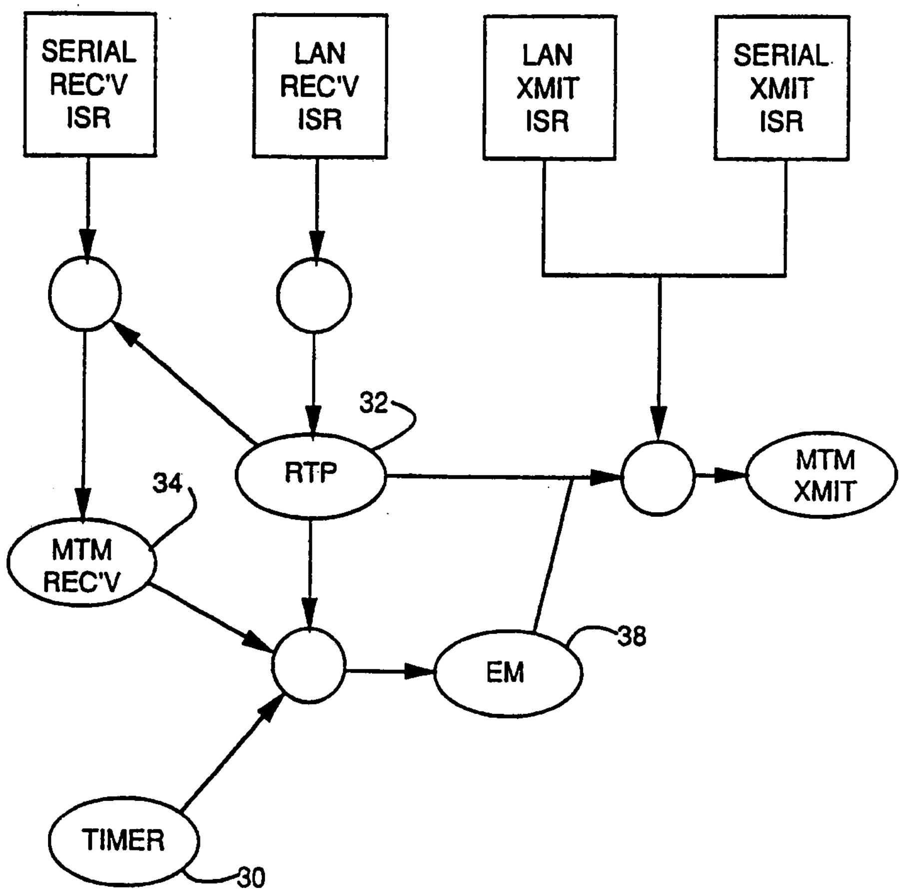
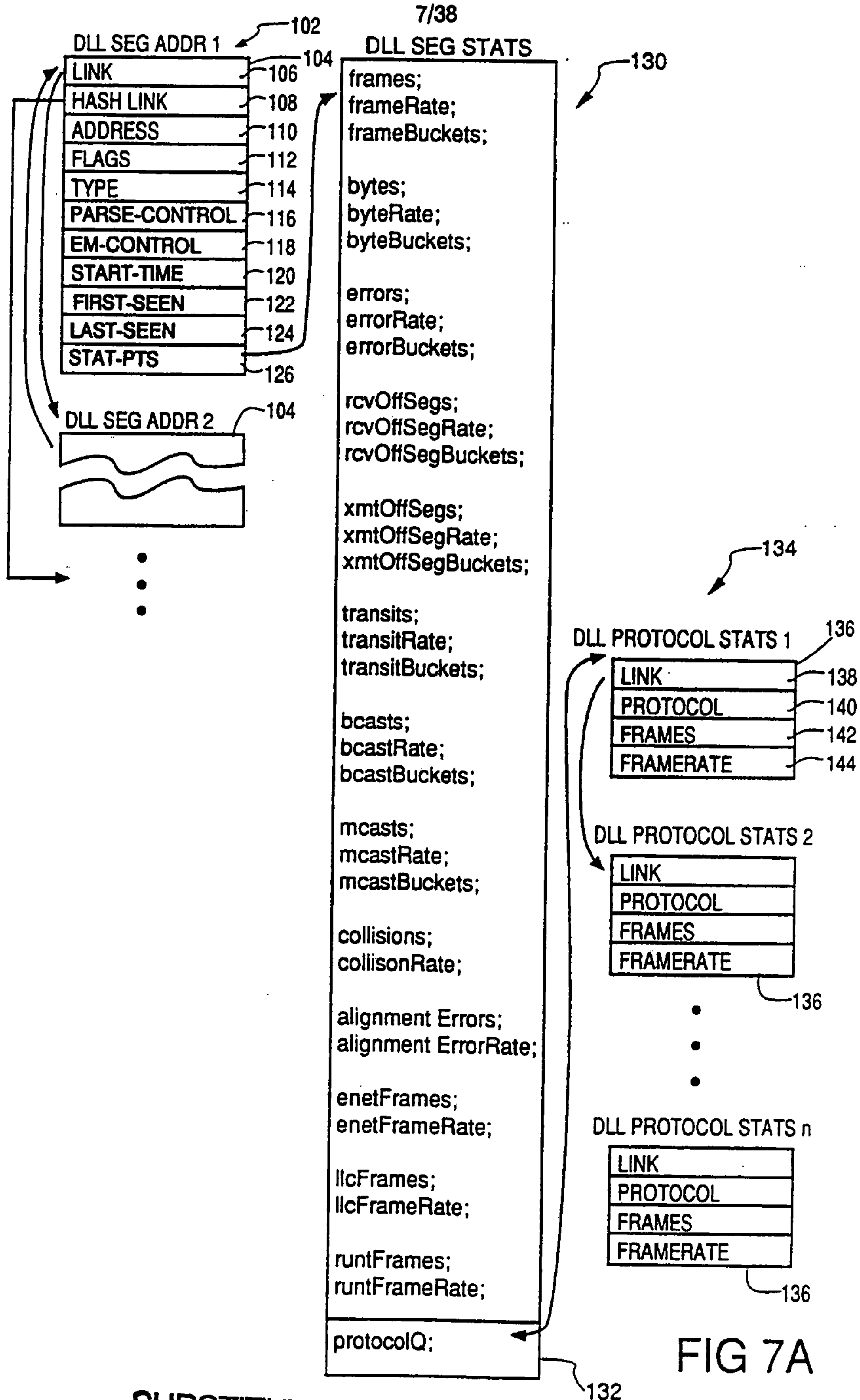
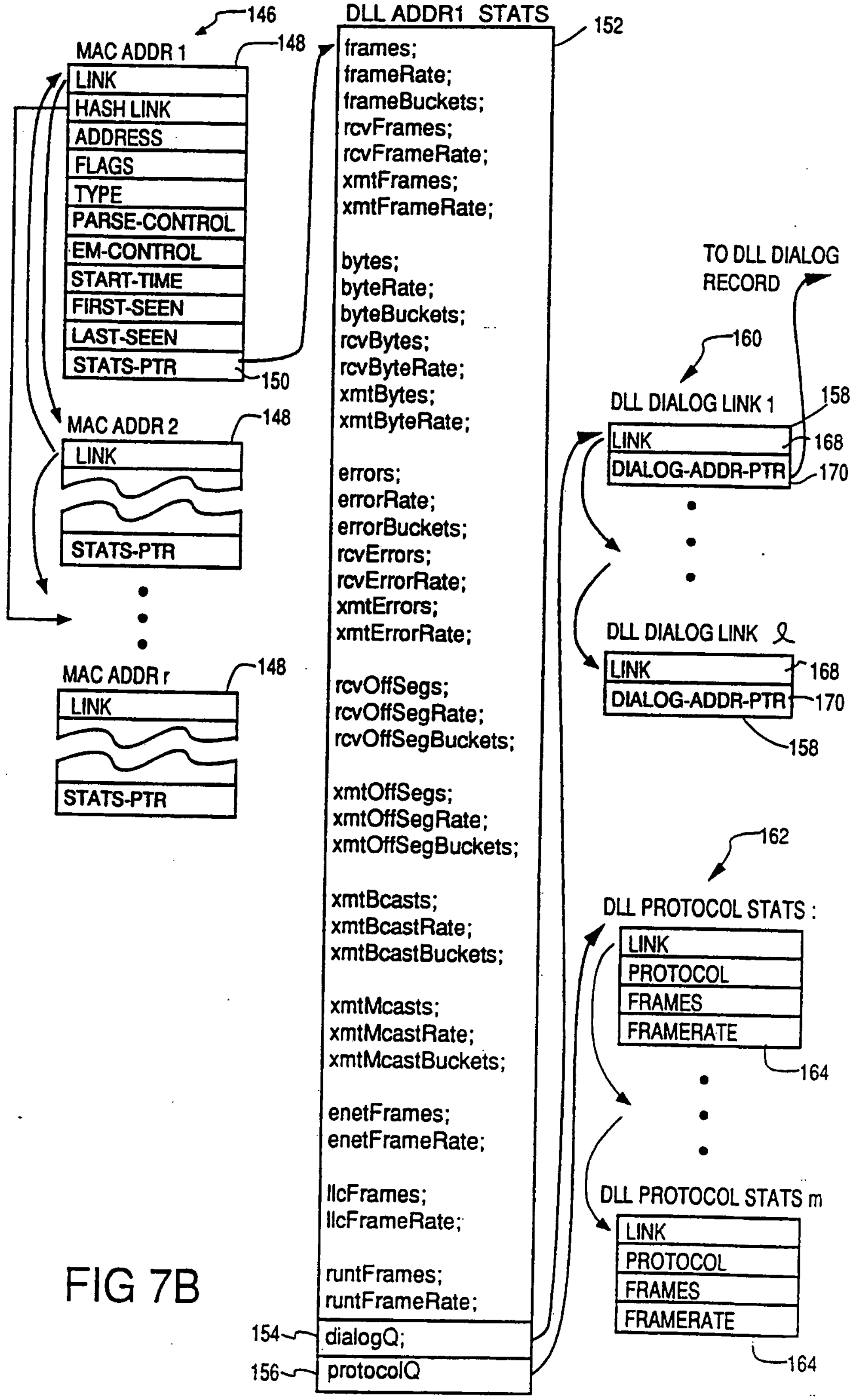

SERIAL REC'V ISR
LAN REC'V ISR
LAN XMIT ISR
SERIAL XMIT ISR
32
RTP
34
MTM XMIT
MTM REC'V
EM
38
TIMER
30

FIG 6



102
DLL SEG ADDR 1
104
LINK
106
HASH LINK
108
ADDRESS
110
FLAGS
112
TYPE
114
PARSE-CONTROL
116
EM-CONTROL
118
START-TIME
120
FIRST-SEEN
122
LAST-SEEN
124
STAT-PTS
126
DLL SEG ADDR 2
104
DLL SEG STATS
130
frames;
frameRate;
frameBuckets;
bytes;
byteRate;
byteBuckets;
errors;
errorRate;
errorBuckets;
rcvOffSegs;
rcvOffSegRate;
rcvOffSegBuckets;
xmtOffSegs;
xmtOffSegRate;
xmtOffSegBuckets;
transits;
transitRate;
transitBuckets;
bcasts;
bcastRate;
bcastBuckets;
mcasts;
mcastRate;
mcastBuckets;
collisions;
collisonRate;
alignment Errors;
alignment ErrorRate;
enetFrames;
enetFrameRate;
llcFrames;
llcFrameRate;
runtFrames;
runtFrameRate;
protocolQ;
132
134
DLL PROTOCOL STATS 1
136
LINK
138
PROTOCOL
140
FRAMES
142
FRAMERATE
144
DLL PROTOCOL STATS 2
LINK
PROTOCOL
FRAMES
FRAMERATE
136
DLL PROTOCOL STATS n
LINK
PROTOCOL
FRAMES
FRAMERATE
136

FIG 7A



146
MAC ADDR 1
148
LINK
HASH LINK
ADDRESS
FLAGS
TYPE
PARSE-CONTROL
EM-CONTROL
START-TIME
FIRST-SEEN
LAST-SEEN
STATS-PTR
150
MAC ADDR 2
148
LINK
STATS-PTR
MAC ADDR r
148
LINK
STATS-PTR
DLL ADDR1 STATS
152
frames;
frameRate;
frameBuckets;
rcvFrames;
rcvFrameRate;
xmtFrames;
xmtFrameRate;
bytes;
byteRate;
byteBuckets;
rcvBytes;
rcvByteRate;
xmtBytes;
xmtByteRate;
errors;
errorRate;
errorBuckets;
rcvErrors;
rcvErrorRate;
xmtErrors;
xmtErrorRate;
rcvOffSegs;
rcvOffSegRate;
rcvOffSegBuckets;
xmtOffSegs;
xmtOffSegRate;
xmtOffSegBuckets;
xmtBcasts;
xmtBcastRate;
xmtBcastBuckets;
xmtMcasts;
xmtMcastRate;
xmtMcastBuckets;
enetFrames;
enetFrameRate;
llcFrames;
llcFrameRate;
runtFrames;
runtFrameRate;
154
dialogQ;
156
protocolQ
TO DLL DIALOG RECORD
160
DLL DIALOG LINK 1
158
LINK
168
DIALOG-ADDR-PTR
170
DLL DIALOG LINK ℓ
LINK
168
DIALOG-ADDR-PTR
170
158
162
DLL PROTOCOL STATS :
LINK
PROTOCOL
FRAMES
FRAMERATE
164
DLL PROTOCOL STATS m
LINK
PROTOCOL
FRAMES
FRAMERATE
164

FIG 7B

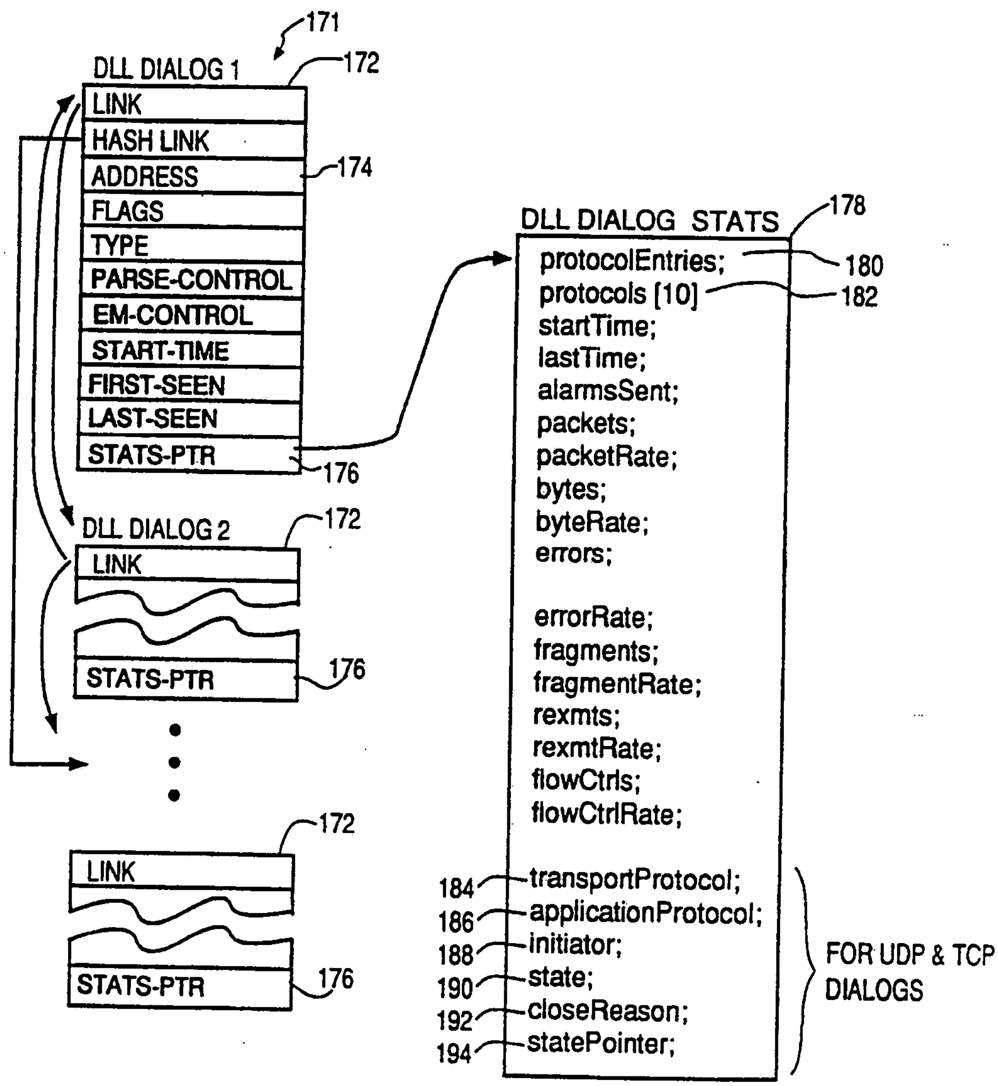

171
DLL DIALOG 1
172
LINK
HASH LINK
ADDRESS
174
FLAGS
TYPE
PARSE-CONTROL
EM-CONTROL
START-TIME
FIRST-SEEN
LAST-SEEN
STATS-PTR
176
DLL DIALOG 2
172
LINK
STATS-PTR
176
172
LINK
STATS-PTR
176
DLL DIALOG STATS
178
protocolEntries;
180
protocols [10]
182
startTime;
lastTime;
alarmsSent;
packets;
packetRate;
bytes;
byteRate;
errors;
errorRate;
fragments;
fragmentRate;
rexmts;
rexmtRate;
flowCtrls;
flowCtrlRate;
184
transportProtocol;
186
applicationProtocol;
188
initiator;
190
state;
192
closeReason;
194
statePointer;
FOR UDP & TCP DIALOGS

FIG 7C

EVENT \ STATE	UNKNOWN	CONNECTING	DATA	CLOSING	CLOSED	INACTIVE
UNKNOWN					S= UNKNOWN AFTER CLOSE ++	S= UNKNOWN
CONNECT REQ OR CONNECT CNF (E.G. TCP SYN)	S= CONNECTING	CONNECTION RETRY ++	S= UNKNOWN OUT OF ORDER++ ACTIVE CONN++	S= UNKNOWN OUT OF ORDER++	S=CONNECTING AFTER CLOSE ++	S=CONNECTING
ABORT (E.G. TCP RST)	S= CLOSED START CLOSE TIMER	S= CLOSED FAILED CONN ++ START CLOSE TIMER	S= CLOSED ACTIVE CONN - - START CLOSE TIMER	S= CLOSED ACTIVE CONN - - START CLOSE TIMER	AFTER CLOSE ++	S= CLOSED START CLOSE TIMER
DATA ACK (E.G. TCP ACK)	LOOK FOR DATA STATE	LOOK FOR DATA STATE	LOOK AT HISTORY	LOOK AT HISTORY	S= UNKNOWN AFTER CLOSE ++	S= UNKNOWN LOOK FOR DATA STATE
RELEASE REQ OR CELEASE CNF (E.G. TCP FIN)	S= CLOSING START CLOSE TIMER	S= CLOSING START CLOSE TIMER	S= CLOSING ACTIVE CONN - - START CLOSE TIMER		S= UNKNOWN AFTER CLOSE ++	S= CLOSING
DATA	LOOK FOR DATA STATE	LOOK FOR DATA STATE	LOOK AT HISTORY	OUT OF ORDER++	S= UNKNOWN AFTER CLOSE ++	S= UNKNOWN LOOK FOR DATA STATE
CLOSE TIMER EXPIRES				S= CLOSED		
INACTIVE TIMER EXPIRES	RECYCLE RESOURCES	RECYCLE RESOURCES FAILED CONN++	RECYCLE RESOURCES ACTIVE CONN --	RECYCLE RESOURCES	RECYCLE RESOURCES	RECYCLE RESOURCES

FIG 8

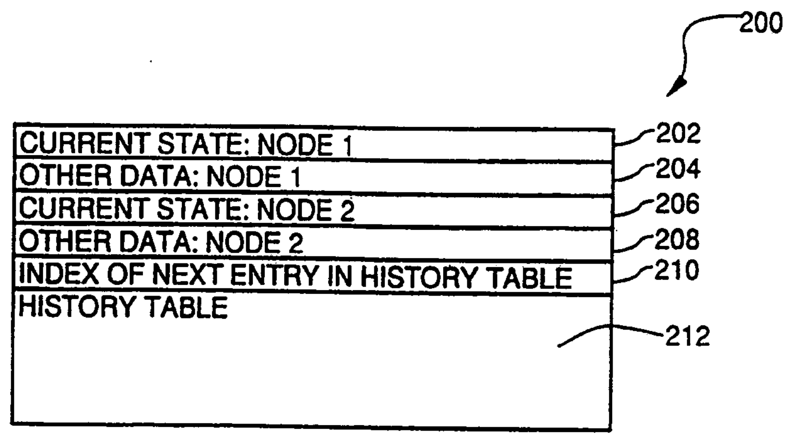

200
CURRENT STATE: NODE 1
202
OTHER DATA: NODE 1
204
CURRENT STATE: NODE 2
206
OTHER DATA: NODE 2
208
INDEX OF NEXT ENTRY IN HISTORY TABLE
210
HISTORY TABLE
212

FIG 9A

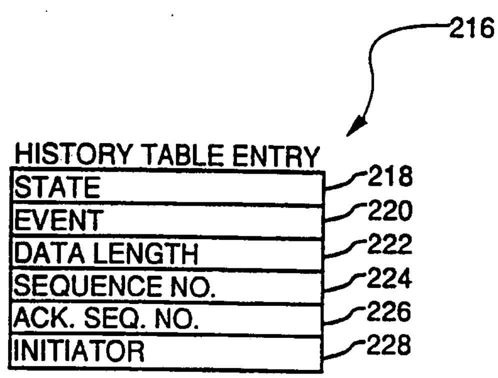

216
HISTORY TABLE ENTRY
STATE
218
EVENT
220
DATA LENGTH
222
SEQUENCE NO.
224
ACK. SEQ. NO.
226
INITIATOR
228

FIG 9B

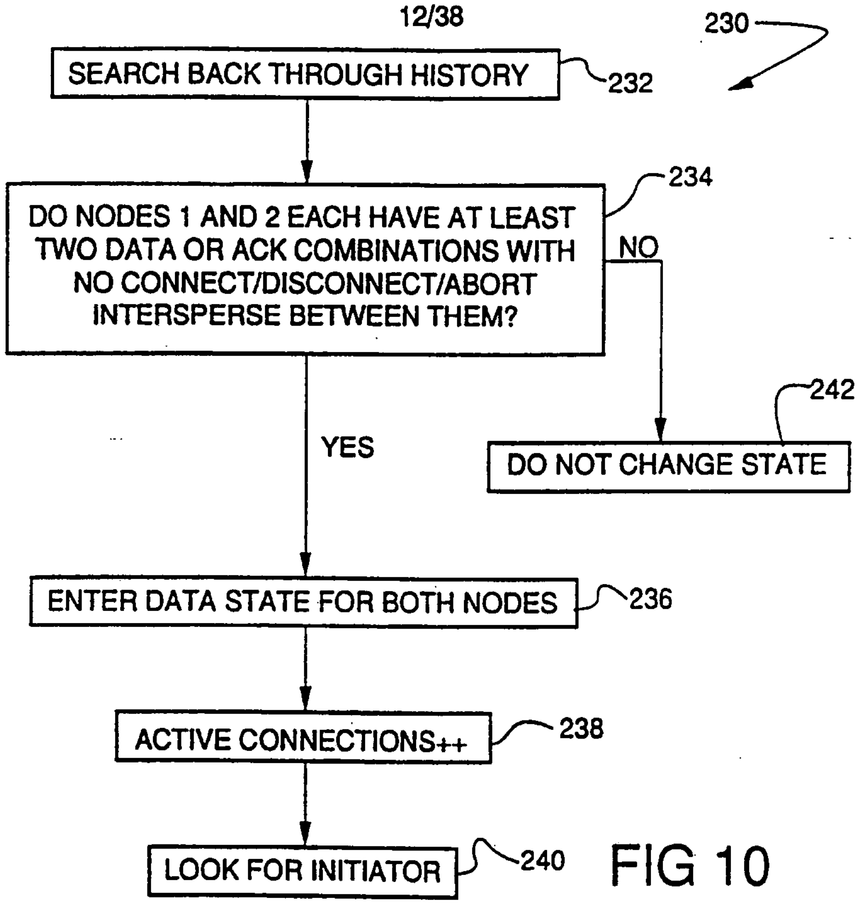

230
SEARCH BACK THROUGH HISTORY
232
DO NODES 1 AND 2 EACH HAVE AT LEAST TWO DATA OR ACK COMBINATIONS WITH NO CONNECT/DISCONNECT/ABORT INTERSPERSE BETWEEN THEM?
234
NO
242
DO NOT CHANGE STATE
YES
ENTER DATA STATE FOR BOTH NODES
236
ACTIVE CONNECTIONS++
238
LOOK FOR INITIATOR
240

FIG 10

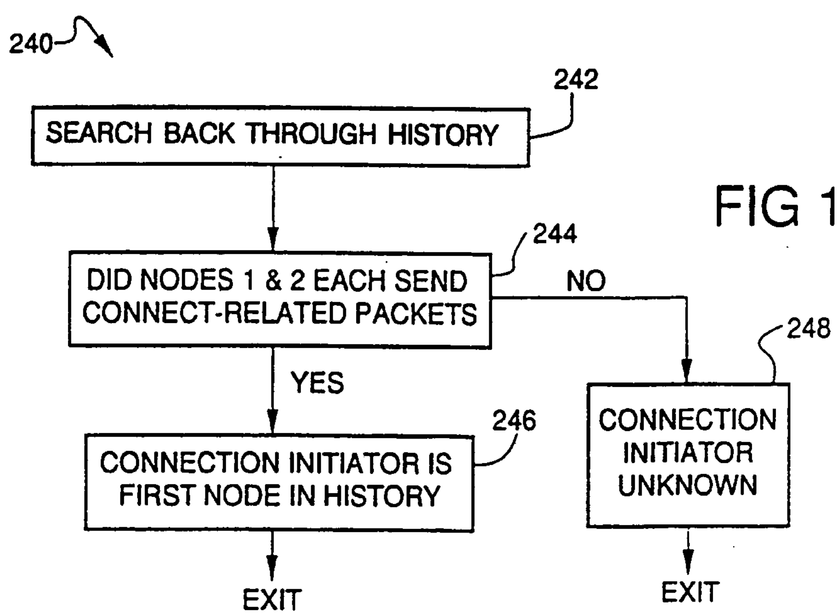

240
242
SEARCH BACK THROUGH HISTORY
244
DID NODES 1 & 2 EACH SEND CONNECT-RELATED PACKETS
NO
248
YES
246
CONNECTION INITIATOR IS FIRST NODE IN HISTORY
CONNECTION INITIATOR UNKNOWN
EXIT
EXIT

FIG 11

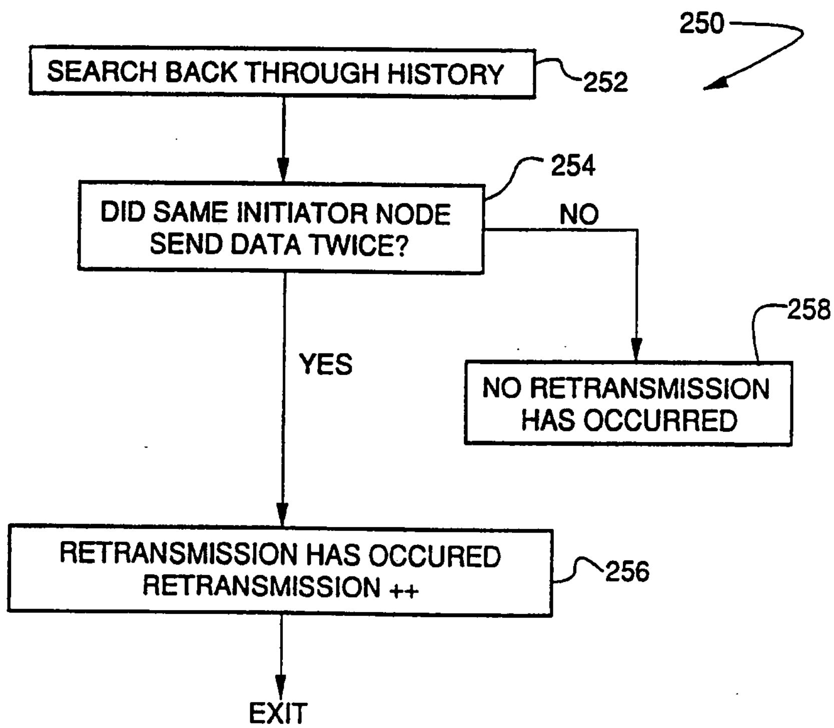

250
SEARCH BACK THROUGH HISTORY
252
DID SAME INITIATOR NODE SEND DATA TWICE?
254
NO
YES
NO RETRANSMISSION HAS OCCURRED
258
RETRANSMISSION HAS OCCURED RETRANSMISSION ++
256
EXIT

FIG 12

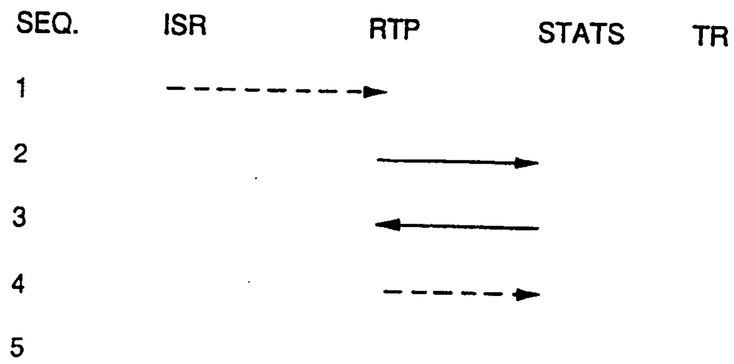

SEQ.
ISR
RTP
STATS
TR
1
2
3
4
5

FIG 13

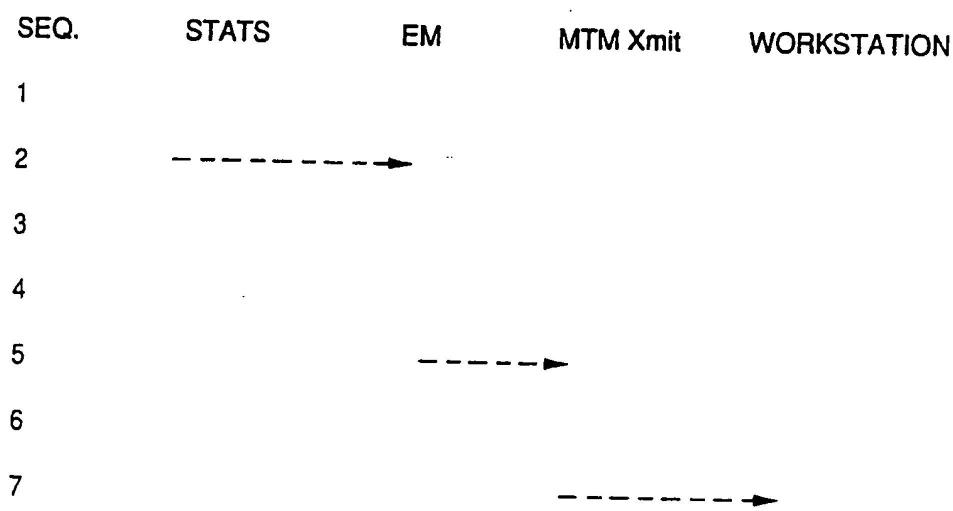

SEQ.
STATS
EM
MTM Xmit
WORKSTATION
1
2
3
4
5
6
7

FIG 14

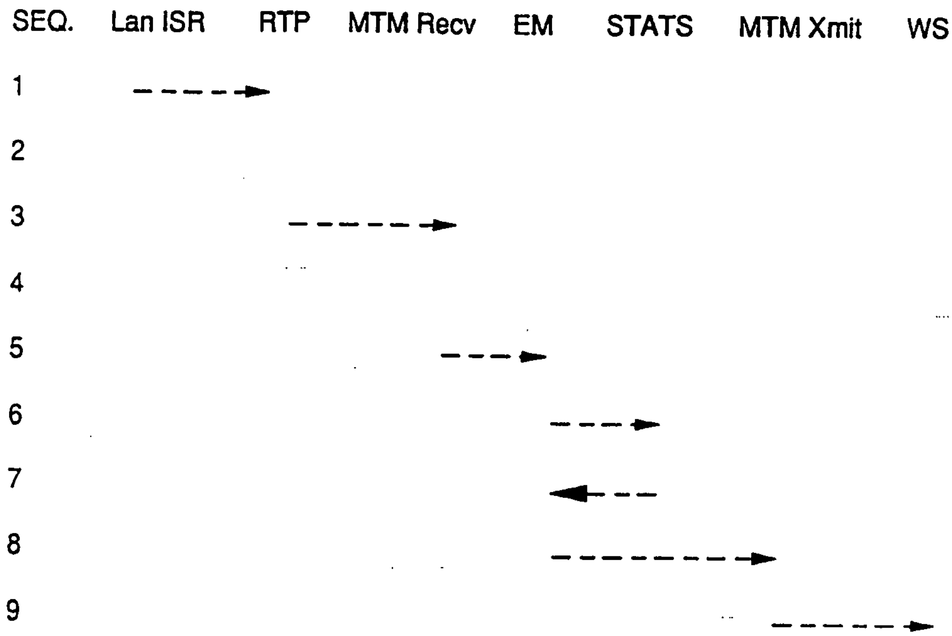

SEQ.
Lan ISR
RTP
MTM Recv
EM
STATS
MTM Xmit
WS
1
2
3
4
5
6
7
8
9

FIG 15


SEQ.
Lan ISR
RTP
MTM Recv
EM
CONTROL
MTM Xmit
WS
1
2
3
4
5
6
7
8
9

FIG 16

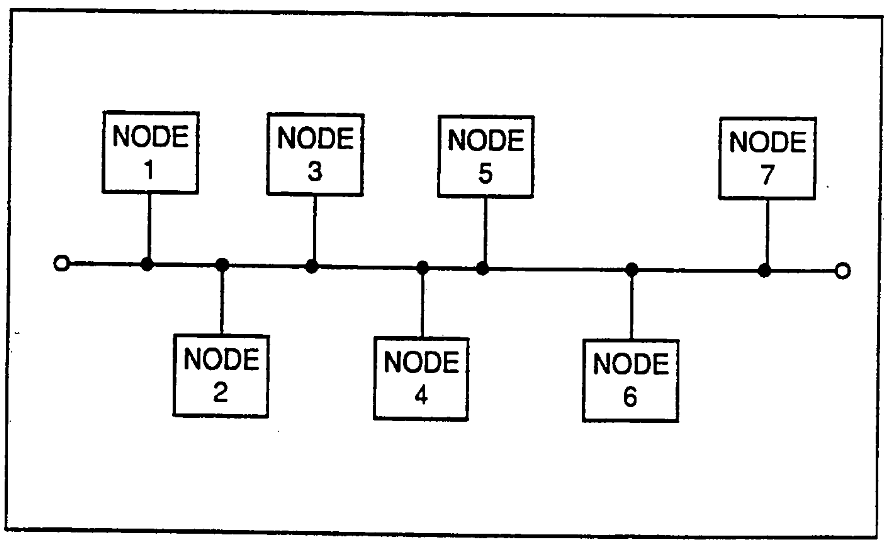


FIG 17

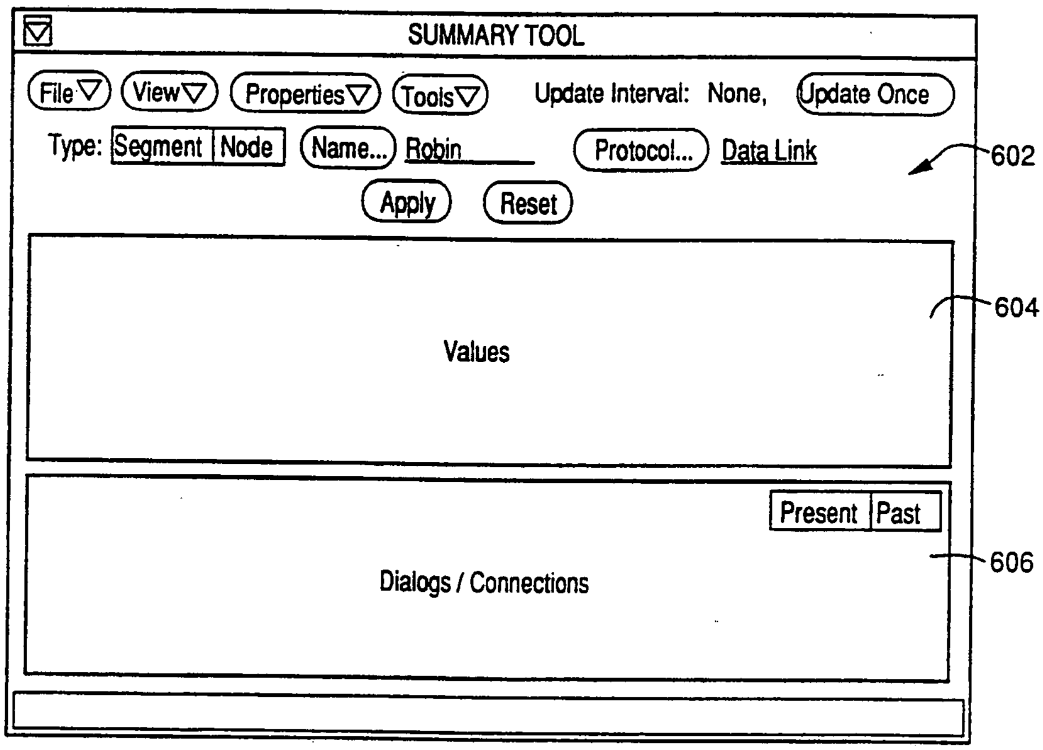

SUMMARY TOOL
File
View
Properties
Tools
Update Interval: None,
Update Once
Type: Segment Node
Name...
Robin
Protocol...
Data Link
Apply
Reset
Values
Present Past
Dialogs / Connections
602
604
606

FIG 18

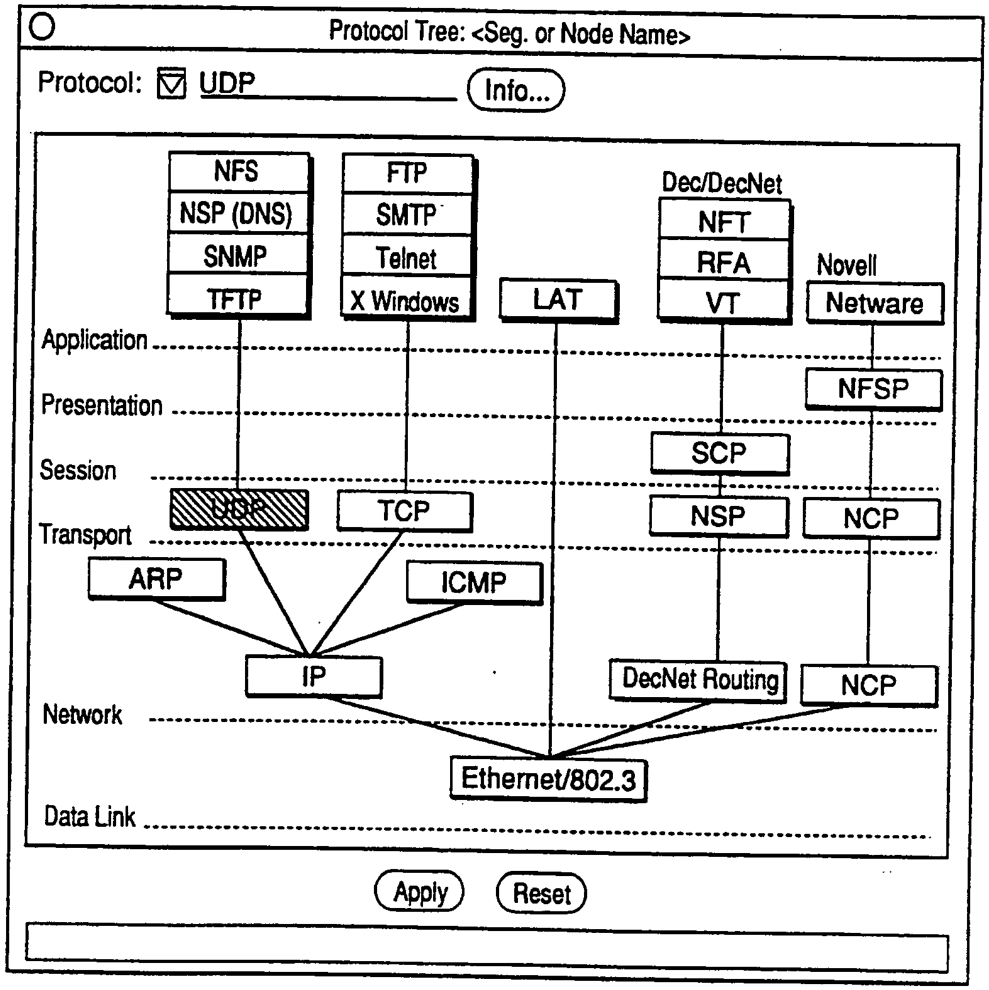

Protocol Tree: <Seg. or Node Name>
Protocol: UDP
Info...
NFS
NSP (DNS)
SNMP
TFTP
FTP
SMTP
Telnet
X Windows
LAT
Dec/DecNet
NFT
RFA
VT
Novell
Netware
Application
Presentation
NFSP
Session
SCP
UDP
TCP
NSP
NCP
Transport
ARP
ICMP
IP
DecNet Routing
NCP
Network
Ethernet/802.3
Data Link
Apply
Reset

FIG 19

Data Link

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Frame Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Broadcast Frm. Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Multicast Frm. Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Off Segment Frames						
In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Protocols (Frm. Rate)		Most Active Nodes (Frm. Rate)	
1234567890123456	nnn %	1234567890123456	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %

Total Segment Bandwidth: nnn % Total Active Dialogs: nn, nnn

FIG 20A

IP

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Broadcast Pkt. Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Multicast Pkt. Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Flow Controls:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Fragments:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Off Segment Packets						
In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Protocols (Pkt. Rate)		Most Active Nodes (Pkt. Rate)	
1234567890123456	nnn %	1234567890123456	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %

Total Segment Bandwidth: nnn % Total Active Dialogs: nn, nnn

FIG 20B

UDP

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Flow Controls:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn

Off Segment Packets

In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Protocols (Pkt. Rate)		Most Active Nodes (Pkt. Rate)	
1234567890123456	nnn %	1234567890123456	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %

Total Segment Bandwidth: nnn %

Total Active Dialogs: nn, nnn

FIG 20C

TCP

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Flow Controls:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Retransmissions:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn

Off Segment Packets

In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Protocols (Pkt. Rate)		Most Active Nodes (Pkt. Rate)	
1234567890123456	nnn %	1234567890123456	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %
<protocol>	nnn %	<node>	nnn %

Total Segment Bandwidth: nnn %

Total Active Connections: nn, nnn

FIG 20D

ICMP

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Off Segment Packets						
In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

ICMP Types Seen (Count)

Address Mask:	nnn,nnn	Redirect:	nnn,nnn
Dst. Unreachable:	nnn,nnn	Source Quench:	nnn,nnn
Echo:	nnn,nnn	Time Exceeded:	nnn,nnn
Param. Problem:	nnn,nnn	Time Stamp:	nnn,nnn

Most Active Nodes (Pkt. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

FIG 20E

NFS

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Flow Controls:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Off Segment Packets						
In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Nodes (Pkt. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

Total Active Dialogs: nn, nnn

FIG 20F

Arp/Rarp

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Off Segment Packets						
In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Nodes (Pkt. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

FIG 20G

Start Time	Last Seen	Dir.	Partner Node	Protocols	Rate	Packets Summary %	Count	Errors
hh:mm:ss	hh:mm:ss	1234	1234567890123456	1234567890123456	nn,nnn /s	nnn %	n,nnn,nnn	nnn,nnn
10:23:04	15:31:47	To	robin	XNS,XEROX-PUP	325 /s	6%	2,641	0
07:21:38	13:25:51	From	hawk	DOD-IP, X25 BBN-SIMNET	87 /s	3%	127	1
10/31/90	08:22:30	?	hawk	APPLETALK	13 /s	1%	24,192	4

FIG 21

23/38

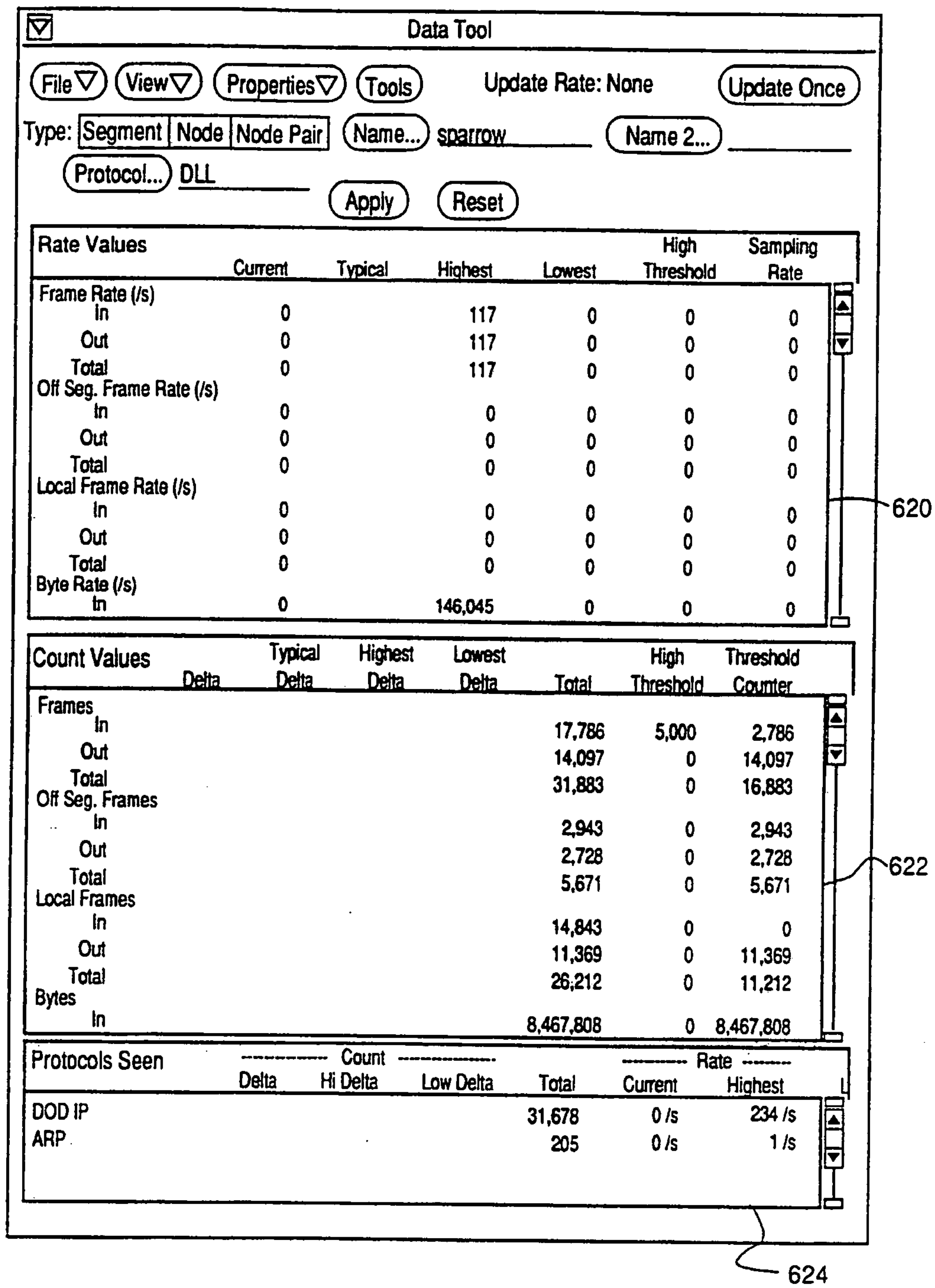


FIG 22

SUBSTITUTE SHEET

628

	Seg1	Seg2	Seg3	....	Segn
Seg1		frame byte error	frame byte error		frame byte error
Seg2	frame byte error		frame byte error		frame byte error
Seg3	frame byte error	frame byte error			frame byte error
⋮					
Segn	frame byte error	frame byte error	frame byte error	....	

FIG 23

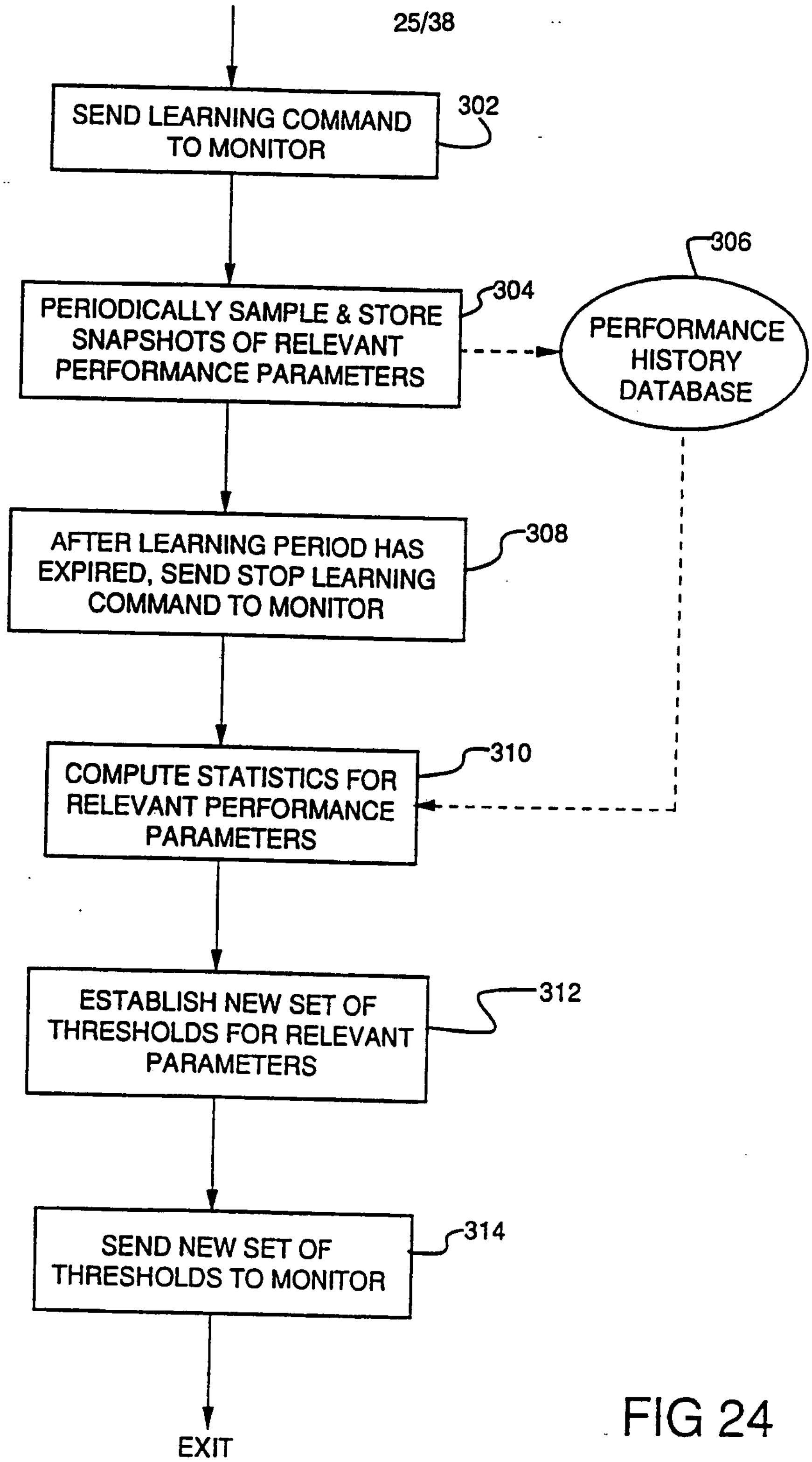
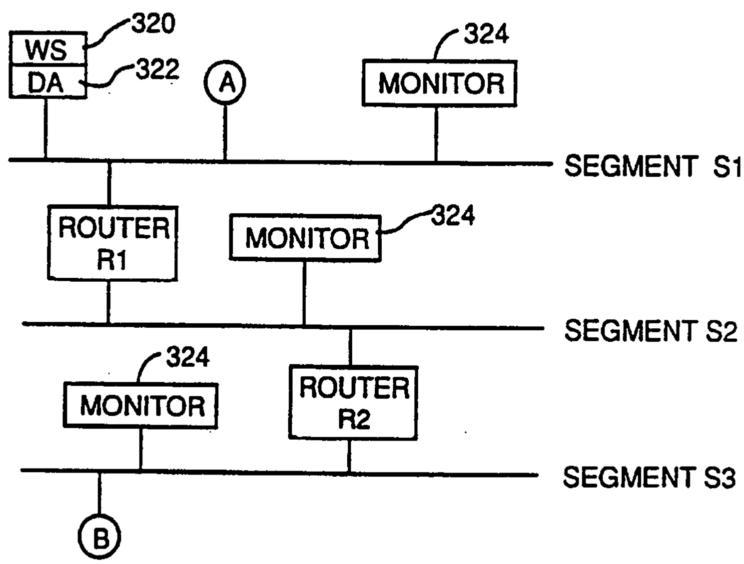

SEND LEARNING COMMAND TO MONITOR
302
PERIODICALLY SAMPLE & STORE SNAPSHOTS OF RELEVANT PERFORMANCE PARAMETERS
304
PERFORMANCE HISTORY DATABASE
306
AFTER LEARNING PERIOD HAS EXPIRED, SEND STOP LEARNING COMMAND TO MONITOR
308
COMPUTE STATISTICS FOR RELEVANT PERFORMANCE PARAMETERS
310
ESTABLISH NEW SET OF THRESHOLDS FOR RELEVANT PARAMETERS
312
SEND NEW SET OF THRESHOLDS TO MONITOR
314
EXIT

FIG 24



320
WS
DA
322
A
324
MONITOR
SEGMENT S1
ROUTER
R1
MONITOR
324
SEGMENT S2
324
MONITOR
ROUTER
R2
SEGMENT S3
B

FIG 25

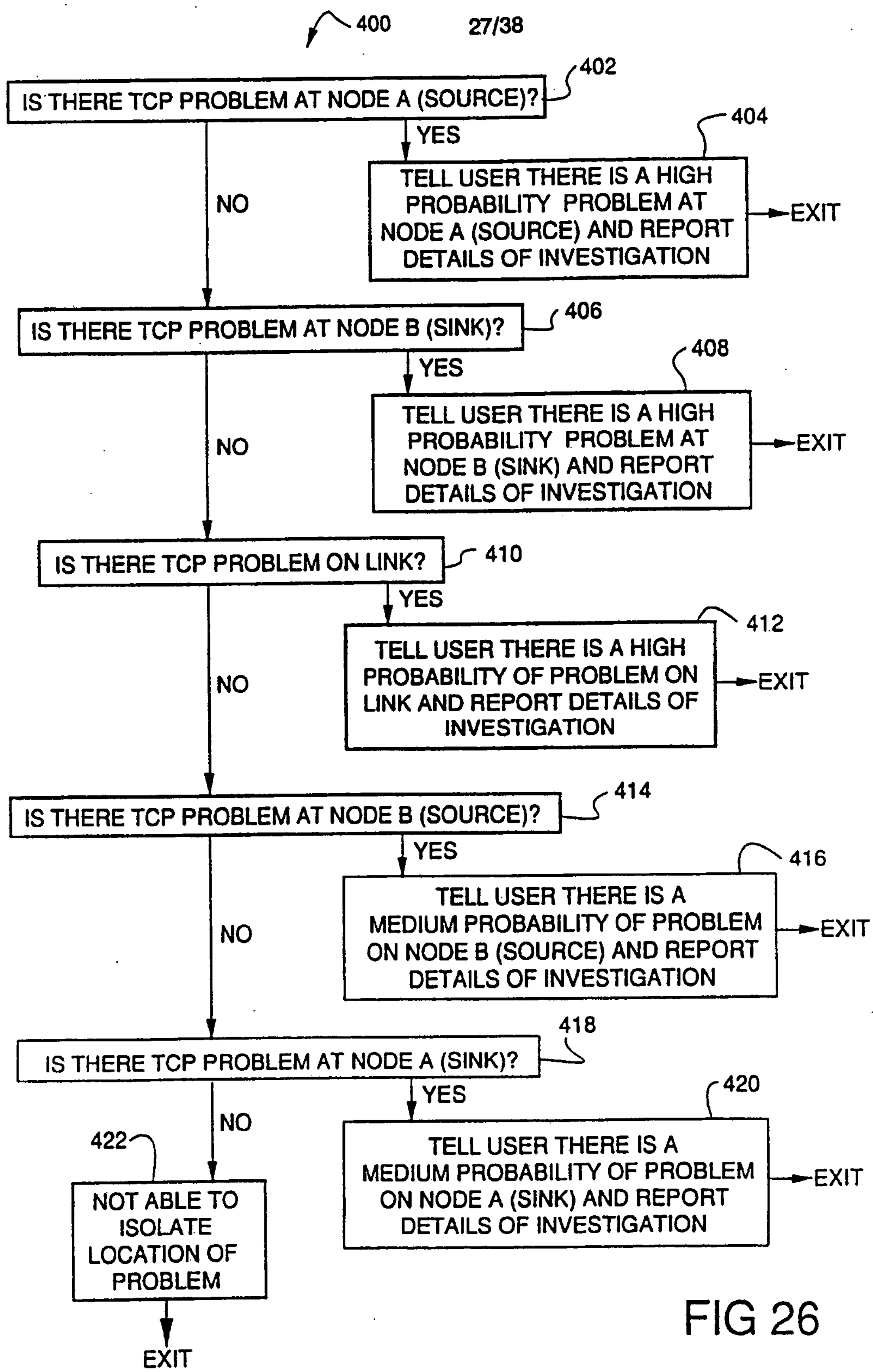

400
IS THERE TCP PROBLEM AT NODE A (SOURCE)?
402
YES
404
TELL USER THERE IS A HIGH PROBABILITY PROBLEM AT NODE A (SOURCE) AND REPORT DETAILS OF INVESTIGATION
EXIT
NO
IS THERE TCP PROBLEM AT NODE B (SINK)?
406
YES
408
TELL USER THERE IS A HIGH PROBABILITY PROBLEM AT NODE B (SINK) AND REPORT DETAILS OF INVESTIGATION
EXIT
NO
IS THERE TCP PROBLEM ON LINK?
410
YES
412
TELL USER THERE IS A HIGH PROBABILITY OF PROBLEM ON LINK AND REPORT DETAILS OF INVESTIGATION
EXIT
NO
IS THERE TCP PROBLEM AT NODE B (SOURCE)?
414
YES
416
TELL USER THERE IS A MEDIUM PROBABILITY OF PROBLEM ON NODE B (SOURCE) AND REPORT DETAILS OF INVESTIGATION
EXIT
NO
418
IS THERE TCP PROBLEM AT NODE A (SINK)?
YES
420
TELL USER THERE IS A MEDIUM PROBABILITY OF PROBLEM ON NODE A (SINK) AND REPORT DETAILS OF INVESTIGATION
EXIT
NO
422
NOT ABLE TO ISOLATE LOCATION OF PROBLEM
EXIT

FIG 26

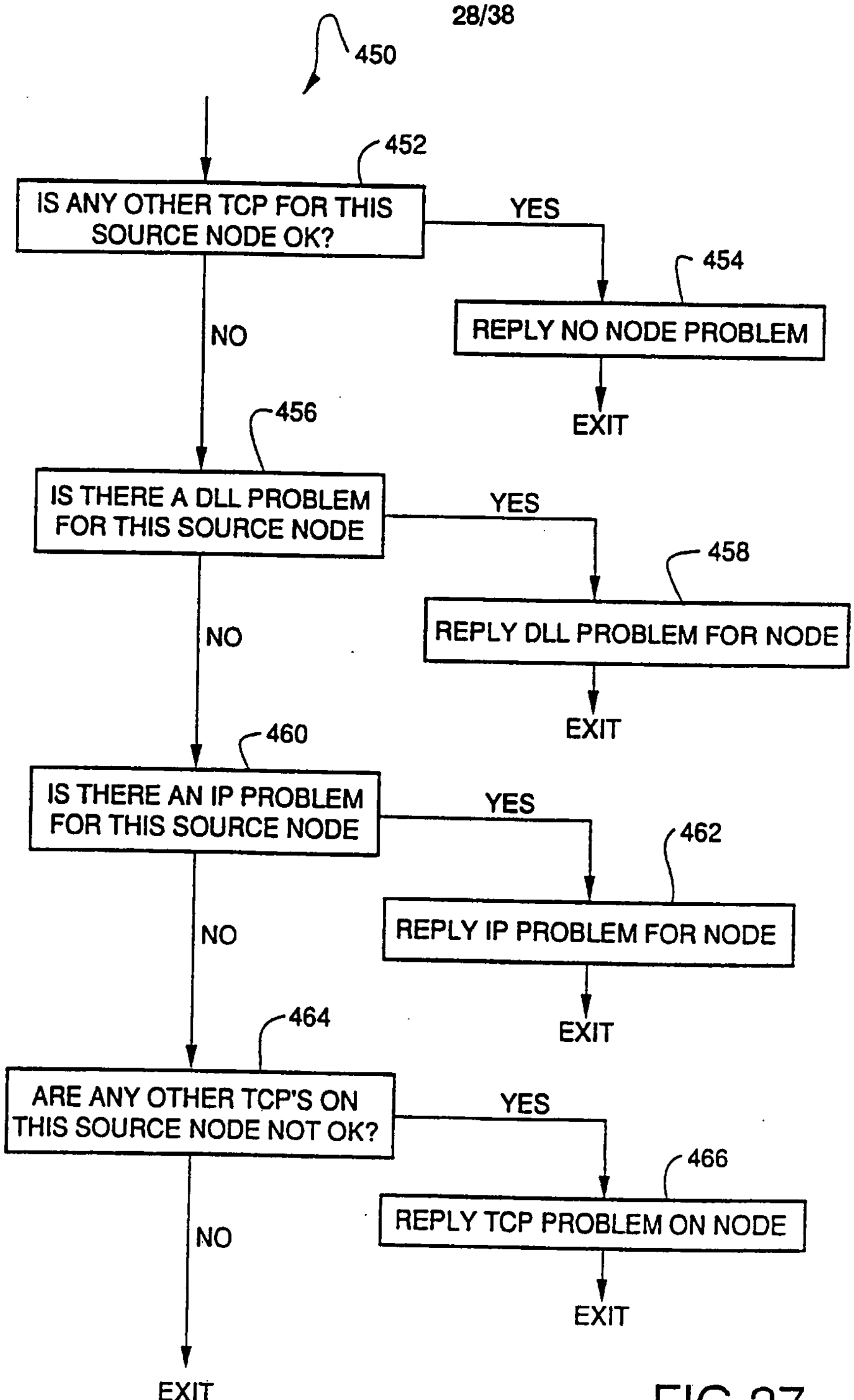

450
452
IS ANY OTHER TCP FOR THIS SOURCE NODE OK?
YES
454
REPLY NO NODE PROBLEM
EXIT
NO
456
IS THERE A DLL PROBLEM FOR THIS SOURCE NODE
YES
458
REPLY DLL PROBLEM FOR NODE
EXIT
NO
460
IS THERE AN IP PROBLEM FOR THIS SOURCE NODE
YES
462
REPLY IP PROBLEM FOR NODE
EXIT
NO
464
ARE ANY OTHER TCP'S ON THIS SOURCE NODE NOT OK?
YES
466
REPLY TCP PROBLEM ON NODE
EXIT
NO
EXIT

FIG 27

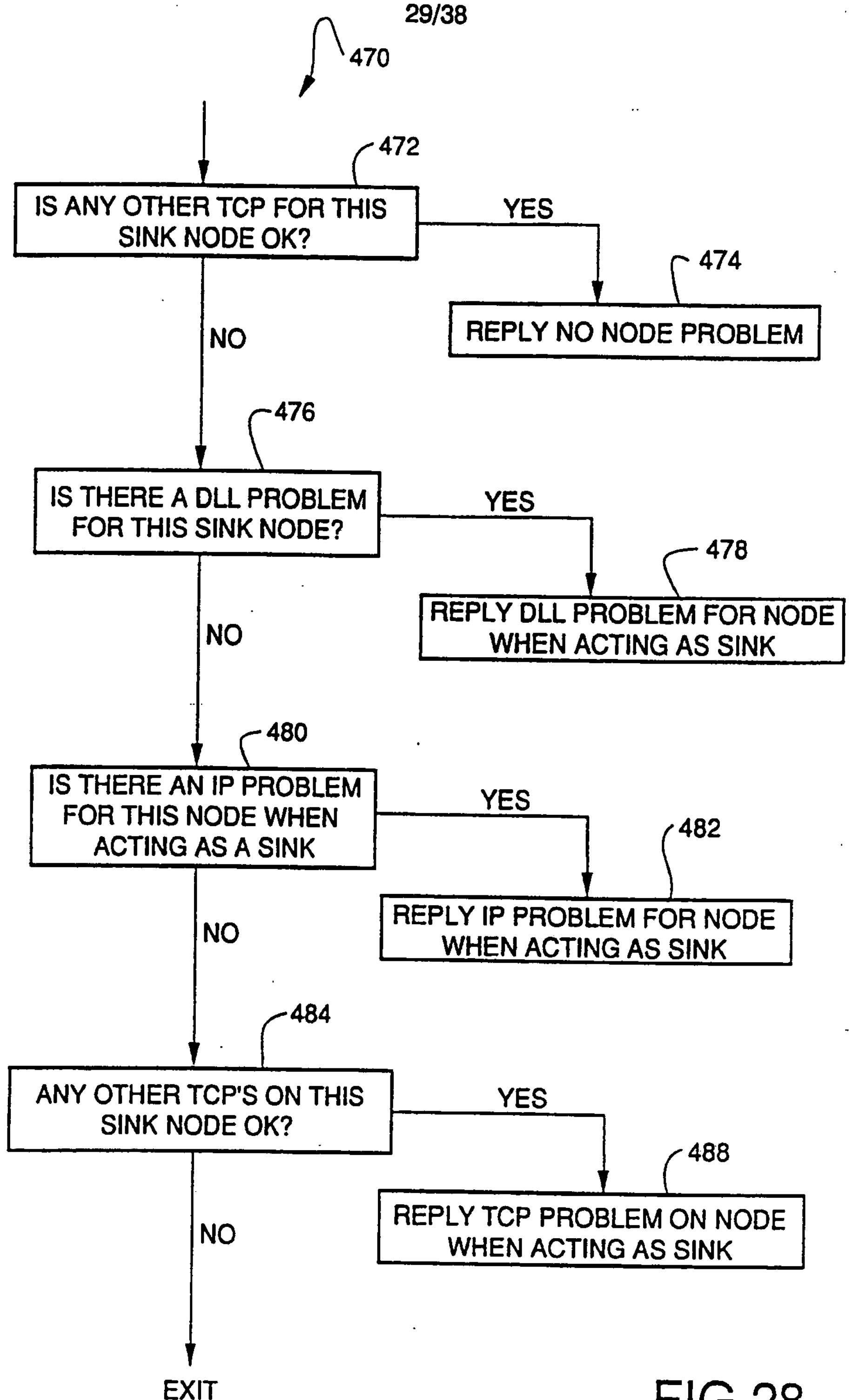
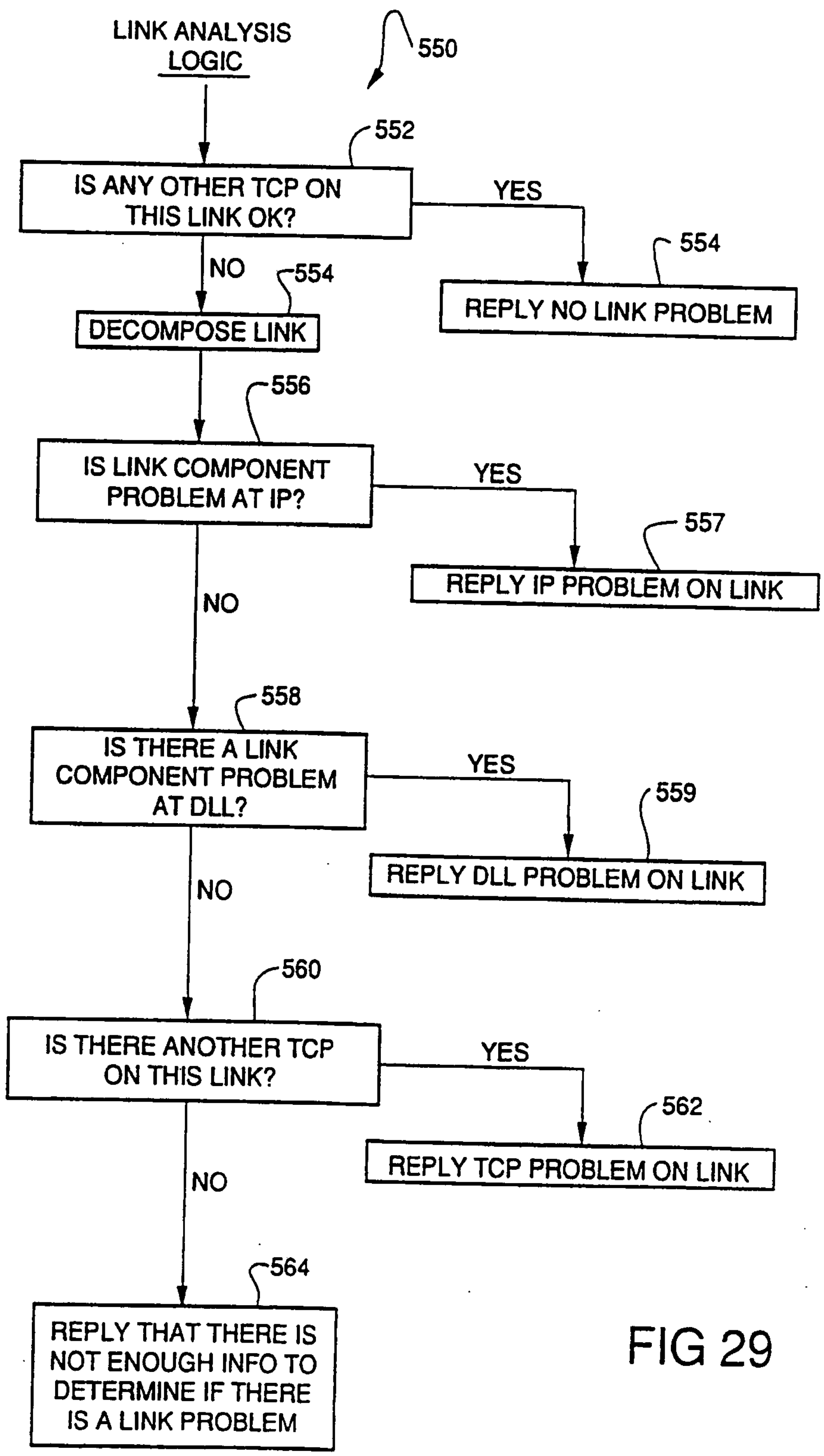

470
472
IS ANY OTHER TCP FOR THIS SINK NODE OK?
YES
474
REPLY NO NODE PROBLEM
NO
476
IS THERE A DLL PROBLEM FOR THIS SINK NODE?
YES
478
REPLY DLL PROBLEM FOR NODE WHEN ACTING AS SINK
NO
480
IS THERE AN IP PROBLEM FOR THIS NODE WHEN ACTING AS A SINK
YES
482
REPLY IP PROBLEM FOR NODE WHEN ACTING AS SINK
NO
484
ANY OTHER TCP'S ON THIS SINK NODE OK?
YES
488
REPLY TCP PROBLEM ON NODE WHEN ACTING AS SINK
NO
EXIT

FIG 28



LINK ANALYSIS LOGIC
550
552
IS ANY OTHER TCP ON THIS LINK OK?
YES
554
REPLY NO LINK PROBLEM
NO
554
DECOMPOSE LINK
556
IS LINK COMPONENT PROBLEM AT IP?
YES
557
REPLY IP PROBLEM ON LINK
NO
558
IS THERE A LINK COMPONENT PROBLEM AT DLL?
YES
559
REPLY DLL PROBLEM ON LINK
NO
560
IS THERE ANOTHER TCP ON THIS LINK?
YES
562
REPLY TCP PROBLEM ON LINK
NO
564
REPLY THAT THERE IS NOT ENOUGH INFO TO DETERMINE IF THERE IS A LINK PROBLEM

FIG 29

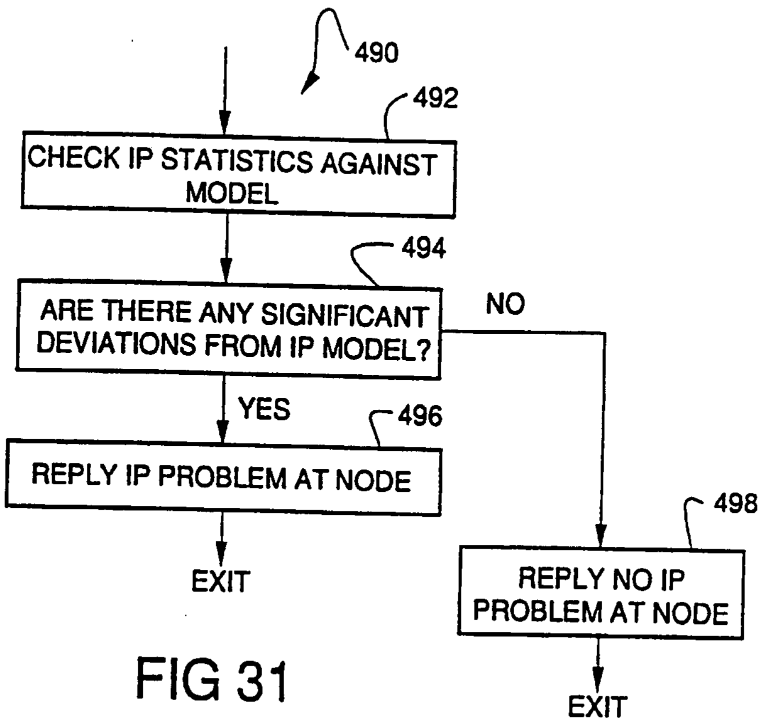

490
492
CHECK IP STATISTICS AGAINST MODEL
494
ARE THERE ANY SIGNIFICANT DEVIATIONS FROM IP MODEL?
NO
YES
496
REPLY IP PROBLEM AT NODE
EXIT
498
REPLY NO IP PROBLEM AT NODE
EXIT

FIG 31

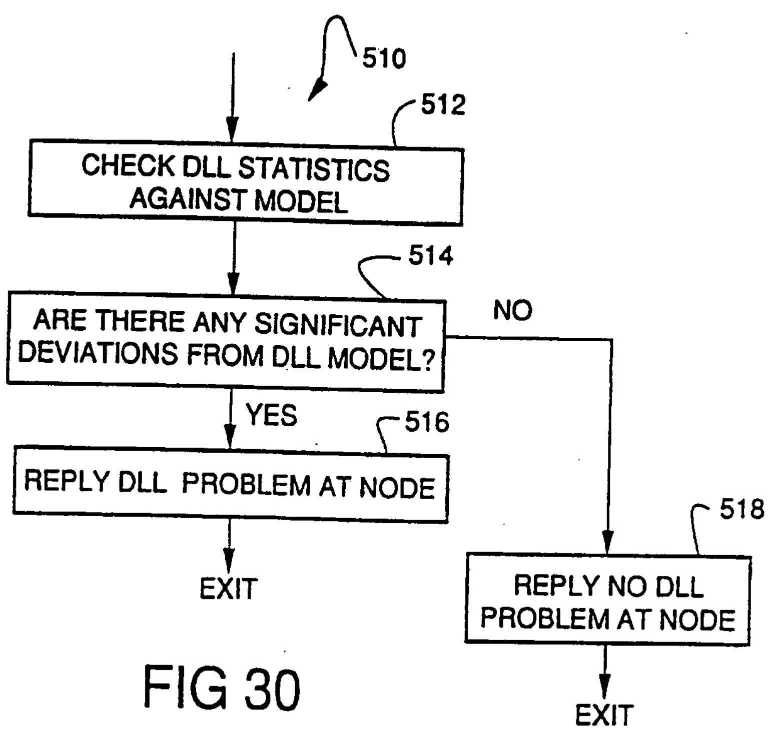

510
512
CHECK DLL STATISTICS AGAINST MODEL
514
ARE THERE ANY SIGNIFICANT DEVIATIONS FROM DLL MODEL?
NO
YES
516
REPLY DLL PROBLEM AT NODE
EXIT
518
REPLY NO DLL PROBLEM AT NODE
EXIT

FIG 30

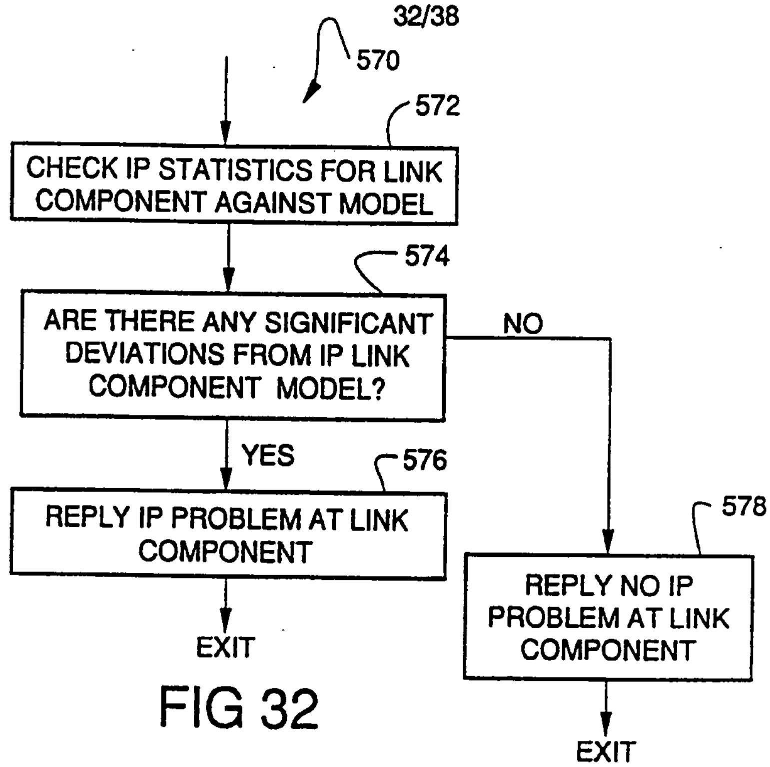

570
572
CHECK IP STATISTICS FOR LINK COMPONENT AGAINST MODEL
574
ARE THERE ANY SIGNIFICANT DEVIATIONS FROM IP LINK COMPONENT MODEL?
NO
YES
576
REPLY IP PROBLEM AT LINK COMPONENT
578
REPLY NO IP PROBLEM AT LINK COMPONENT
EXIT
EXIT

FIG 32

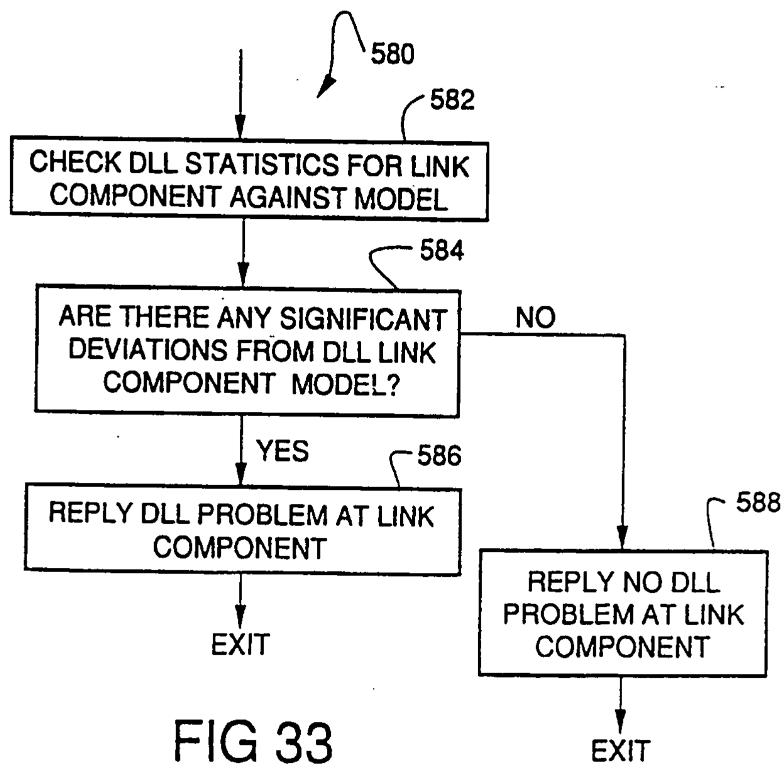

580
582
CHECK DLL STATISTICS FOR LINK COMPONENT AGAINST MODEL
584
ARE THERE ANY SIGNIFICANT DEVIATIONS FROM DLL LINK COMPONENT MODEL?
NO
YES
586
REPLY DLL PROBLEM AT LINK COMPONENT
588
REPLY NO DLL PROBLEM AT LINK COMPONENT
EXIT
EXIT

FIG 33

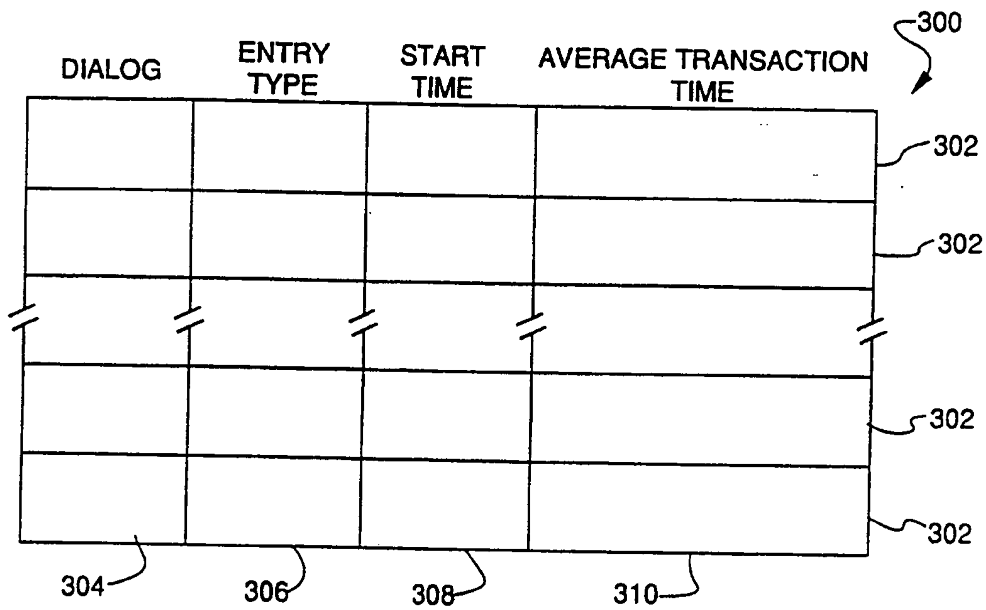

300
DIALOG
ENTRY TYPE
START TIME
AVERAGE TRANSACTION TIME
302
302
302
302
304
306
308
310

FIG 34

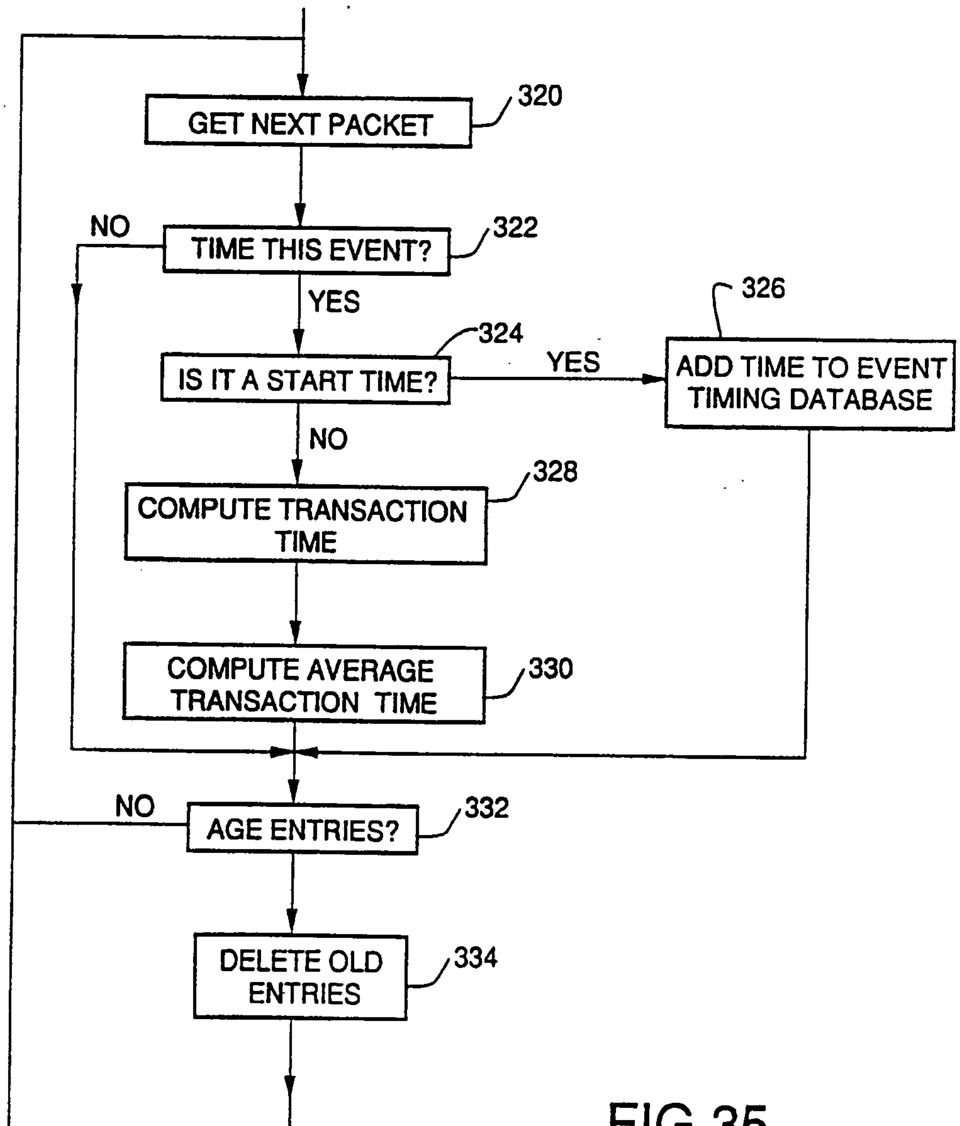

320
GET NEXT PACKET
NO
322
TIME THIS EVENT?
YES
324
IS IT A START TIME?
YES
326
ADD TIME TO EVENT TIMING DATABASE
NO
328
COMPUTE TRANSACTION TIME
330
COMPUTE AVERAGE TRANSACTION TIME
NO
332
AGE ENTRIES?
334
DELETE OLD ENTRIES

FIG 35

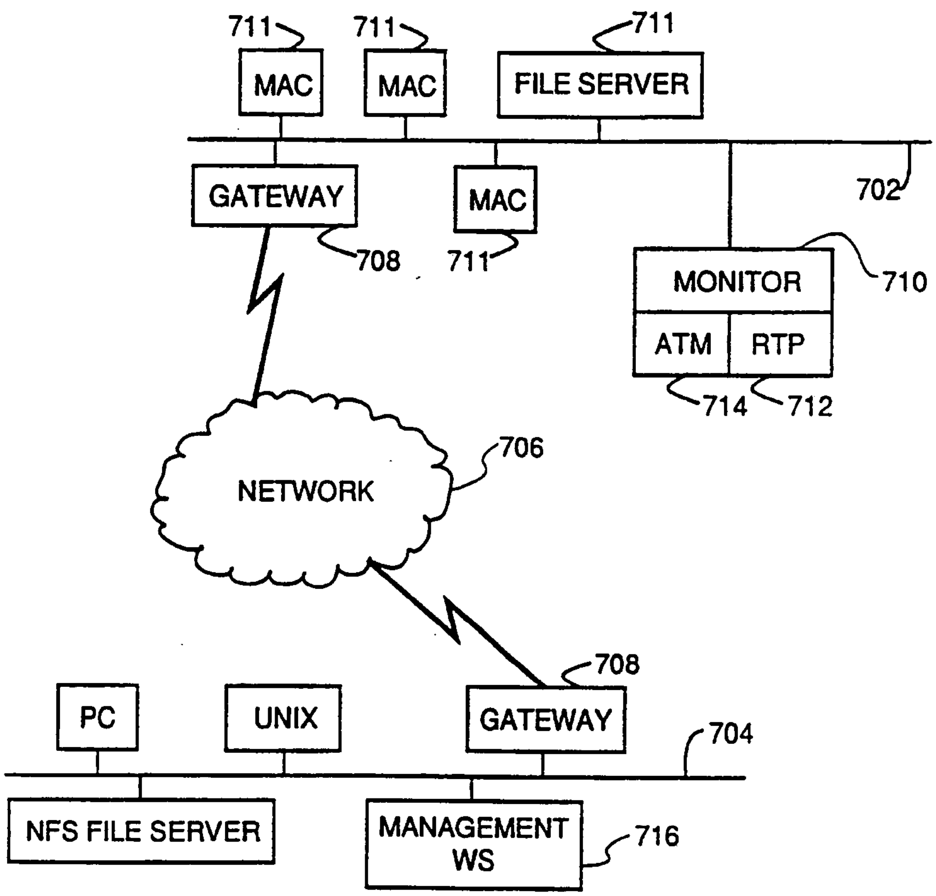

711
711
711
MAC
MAC
FILE SERVER
GATEWAY
MAC
702
708
711
710
MONITOR
ATM
RTP
714
712
706
NETWORK
708
PC
UNIX
GATEWAY
704
NFS FILE SERVER
MANAGEMENT WS
716

FIG 36

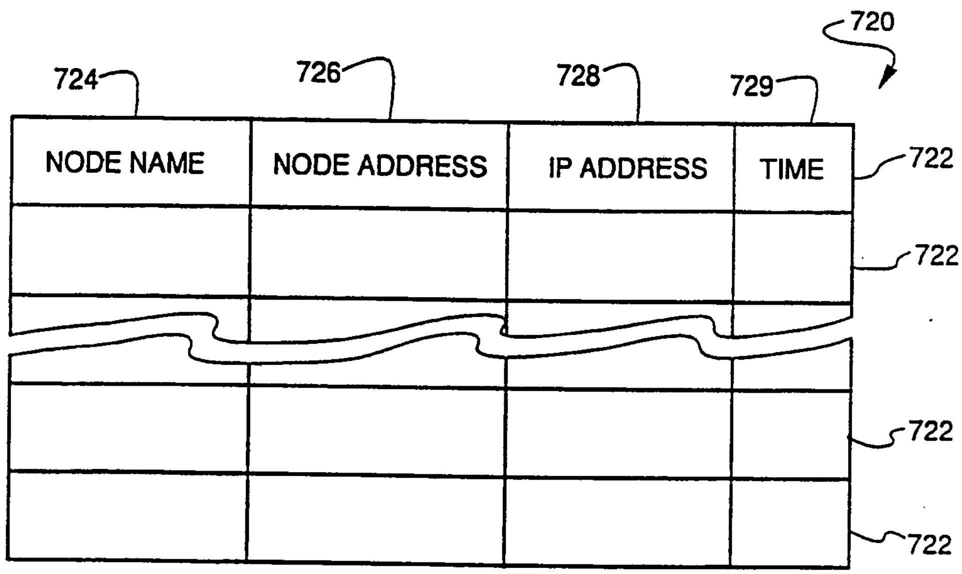

720
724
726
728
729
NODE NAME
NODE ADDRESS
IP ADDRESS
TIME
722
722
722
722

FIG 37

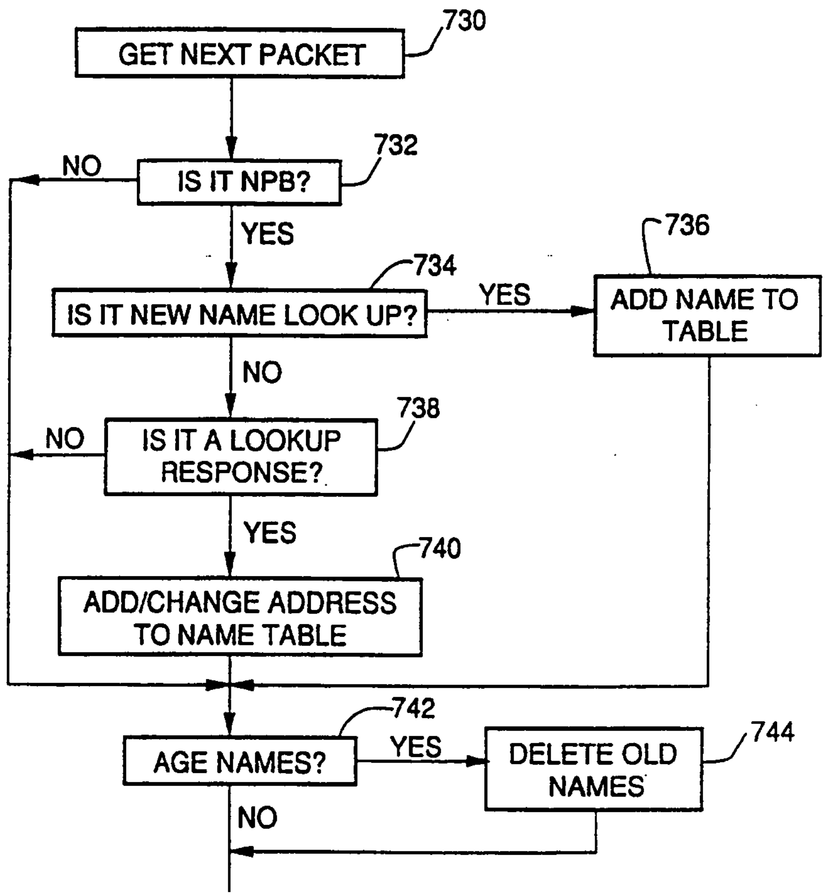

GET NEXT PACKET
730
IS IT NPB?
732
NO
YES
IS IT NEW NAME LOOK UP?
734
YES
ADD NAME TO TABLE
736
NO
IS IT A LOOKUP RESPONSE?
738
NO
YES
ADD/CHANGE ADDRESS TO NAME TABLE
740
AGE NAMES?
742
YES
DELETE OLD NAMES
744
NO

FIG 38

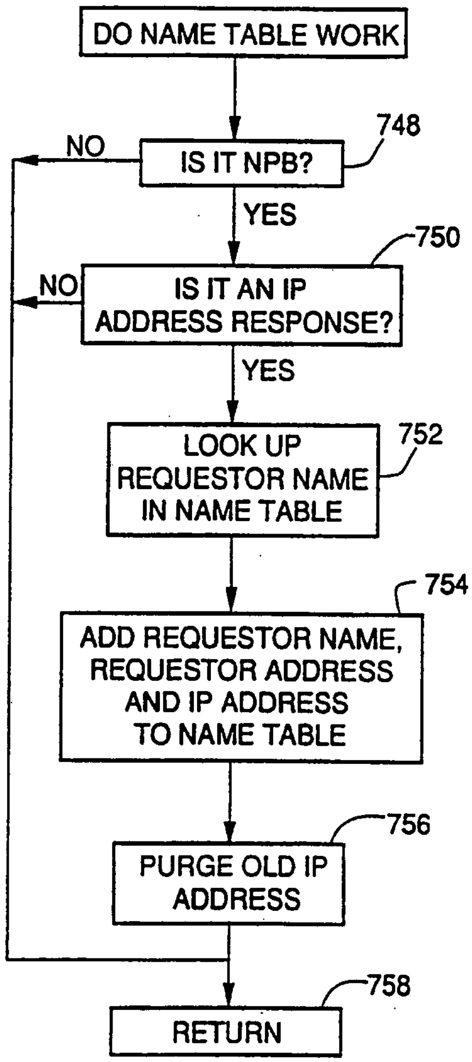

DO NAME TABLE WORK
IS IT NPB?
748
NO
YES
IS IT AN IP ADDRESS RESPONSE?
750
NO
YES
LOOK UP REQUESTOR NAME IN NAME TABLE
752
ADD REQUESTOR NAME, REQUESTOR ADDRESS AND IP ADDRESS TO NAME TABLE
754
PURGE OLD IP ADDRESS
756
RETURN
758

FIG 39

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US92/02995

A. CLASSIFICATION OF SUBJECT MATTER

IPC(5) :H04J 3/14; H04J 3/24; H04L 12/56
US CL :370/13, 17, 94.1; 340/825.52
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 370/60; 371/20.1; 340/825.06, 825.07, 825.53; 364/514, 550

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

USPTO APS (Network Monitor);
(Protocol analyzer)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,P Y	US, A, 5,101,402 (Chiu et al) 31 March 1992 (31.03.92). Column 6, line 32 to column 8, line 10. Figs. 15 and 16.	1-7 24-26
X Y	US, A, 4,887,260 (Carden et al) 12 December 1989 (12.12.89). Column 3, lines 21-51; Column 5, lines 50-68; Column 6, line 48 to column 7, line 38; Fig. 6.	1,3-7,9 24-26,29
A,P	US, A, 5,025,491 (Tsuchiya et al) 18 June 1991 (18.06.91)	30
X	US, A, 4,817,080 (Soha) 28 March 1989 (28.03.89) column 4, lines 23.31; column 5, lines 19-37; claim 1; Fig.1 and 3.	1,24-26

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be part of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

Date of the actual completion of the international search	Date of mailing of the international search report
08 JULY 1992	31 JUL 1992
Name and mailing address of the ISA/ Commissioner of Patents and Trademarks Box PCT	Authorized officer fn. H. KIZOU NGUYEN NGOC-HO INTERNATIONAL DIVISION